INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng 2003; **1**:1–21 *Prepared using* nmeauth.cls *[Version: 2000/01/19 v2.0]*

A Comparison of Eigensolvers for Large-scale 3D Modal Analysis using AMG-Preconditioned Iterative Methods

Peter Arbenz[1], Ulrich L. Hetmaniuk[2], Richard B. Lehoucq[2], Raymond S. Tuminaro[3]

[1] *Swiss Federal Institute of Technology (ETH), Institute of Scientific Computing, CH-8092 Zurich, Switzerland (arbenz@inf.ethz.ch)*
[2] *Sandia National Laboratories, Computational Mathematics & Algorithms, MS 1110, P.O.Box 5800, Albuquerque, NM 87185-1110 (ulhetma@sandia.gov,rblehou@sandia.gov).*
[3] *Sandia National Laboratories, Computational Mathematics & Algorithms, MS 9159, P.O.Box 969, Livermore, CA 94551 (rstumin@sandia.gov).*

SUMMARY

The goal of our paper is to compare a number of algorithms for computing a large number of eigenvectors of the generalized symmetric eigenvalue problem arising from a modal analysis of elastic structures. The shift-invert Lanczos algorithm has emerged as the workhorse for the solution of this generalized eigenvalue problem; however a sparse direct factorization is required for the resulting set of linear equations. Instead, our paper considers the use of preconditioned iterative methods. We present a brief review of available preconditioned eigensolvers followed by a numerical comparison on three problems using a scalable algebraic multigrid (AMG) preconditioner. Copyright © 2003 John Wiley & Sons, Ltd.

KEY WORDS: Eigenvalues, large sparse symmetric eigenvalue problems, modal analysis, algebraic multigrid, preconditioned eigensolvers, shift-invert Lanczos

1. Introduction

The goal of our paper is to compare a number of algorithms for computing a large number of eigenvectors of the generalized eigenvalue problem

$$\mathbf{K}\mathbf{x} = \lambda\mathbf{M}\mathbf{x}, \qquad \mathbf{K}, \mathbf{M} \in \mathbb{R}^{n\times n}, \tag{1}$$

using preconditioned iterative methods. The matrices $\mathbf{K}$ and $\mathbf{M}$ are large, sparse, and symmetric positive definite and they arise in a modal analysis of elastic structures.

*Correspondence to: Rich Lehoucq, Sandia National Laboratories, Computational Mathematics & Algorithms, MS 1110, P.O.Box 5800, Albuquerque, NM 87185-1110

Contract/grant sponsor: The work of Dr. Arbenz was in part supported by the CSRI, Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy; contract/grant number: DE-AC04-94AL85000

Received January 4, 2005
Revised January 4, 2005

The current state of the art is to use a block Lanczos [20] code with a shift-invert transformation $(\mathbf{K} - \sigma\mathbf{M})^{-1}\mathbf{M}$. The resulting set of linear equations is solved by forward and backward substitution with the factors computed by a sparse direct factorization. This algorithm is commercially available and is incorporated in the MSC.Nastran finite element library.

The three major costs associated with a shift-invert block Lanczos code are

- factoring $\mathbf{K} - \sigma\mathbf{M}$;
- solving linear systems with the above factor;
- the cost (and storage) of maintaining the orthogonality of Lanczos vectors.

The block shift-invert Lanczos approach allows for an efficient solution of (1) as long as any of the above three costs (or a combination of them) are not prohibitive. We refer to [20] for further details and information on a state-of-the-art block Lanczos implementation for problems in structural dynamics. However, we note that the factorization costs increase quadratically with the dimension n. Secondly, this Lanczos algorithm contains a scheme for quickly producing a series of shifts $\sigma = \sigma_1, \ldots, \sigma_p$ that extends over the frequency range of interest and that requires further factorizations.

What if performing a series of sparse direct factorizations becomes prohibitively expensive because the dimension n is large, the frequency range of interest is wide, or both cases apply? The goal of our paper will be to investigate how preconditioned iterative methods perform when the dimension n is large (of order $10^5 - 10^6$) and when a large number, say a few hundred eigenvalues and eigenvectors, are to be computed. With the use of an algebraic multigrid (AMG) preconditioner, the approaches we consider are the following:

- replace the sparse direct method with an AMG-preconditioned conjugate gradient iteration within the shift-invert Lanczos algorithm;
- replace the shift-invert Lanczos algorithm with an AMG-preconditioned eigenvalue algorithm.

The former approach is not new and neither are algorithms for the latter alternative (see [26]). What we propose is a comparison of several algorithms on some representative problems in vibrational analysis. Several recent studies have demonstrated the viability of AMG preconditioners for problems in computational mechanics [1, 38] but to the best of our knowledge, there are no comparable studies for vibration analysis.

The paper is organized as follows. We present a general derivation of the algorithms in Section 2 followed by details in Sections 3–4 associated with

- our implementation of LOBPCG [25];
- our block extensions of DACG [7], the Davidson [12] algorithm, and the Jacobi-Davidson variant JDCG [28];
- and our minor modification of the implicitly restarted Lanczos method [35] in ARPACK [27].

We also provide pseudocode for the implementations that we tested. We hope these descriptions prove useful to other researchers (and the authors of the algorithms). Finally, Section 5 presents our numerical experiments that we performed by means of realistic eigenvalue problems stemming from finite element discretizations of elastodynamic problems in structural

dynamics. Although our problems are of most interest for structural analysts, we believe that our results are applicable to problems in other domains such as computational chemistry or electromagnetism provided that they are real-symmetric or Hermitian and a multilevel preconditioner is available.

2. Overview of Algorithms

This Section presents the basic ingredients of the algorithms compared in our paper. We first discuss a generic algorithm that embodies the salient issues of all our algorithms. We then highlight some of the key aspects of the algorithms we compare. The final subsection reviews some useful notation for the pseudocodes provided.

2.1. A generic algorithm

Let the eigenvalues of problem (1) be arranged in ascending order,

$$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n, \tag{2}$$

and let $\mathbf{K}\mathbf{u}_j = \lambda_j \mathbf{M}\mathbf{u}_j$ where the eigenvectors $\mathbf{u}_j$ are assumed to be $\mathbf{M}$-orthonormalized,

$$\langle \mathbf{u}_i, \mathbf{u}_j \rangle = \mathbf{u}_i^T \mathbf{M}\mathbf{u}_j = \delta_{ij}.$$

The algorithms we compare are designed to exploit the characterization of the eigenvalues of (1) as successive minima of the Rayleigh quotient

$$\rho(\mathbf{x}) = \frac{\mathbf{x}^T\mathbf{K}\mathbf{x}}{\mathbf{x}^T\mathbf{M}\mathbf{x}}, \quad \forall\ \mathbf{x} \in \mathbb{R}^n,\ \mathbf{x} \neq \mathbf{0}. \tag{3}$$

Algorithm 2.1 lists the key steps of all the algorithms. Ultimately, the success of an algorithm crucially depends upon the subspace $\mathcal{S}$ constructed. Our generic algorithm is also colloquially

ALGORITHM 2.1: **Generic Eigenvalue Algorithm (outer loop)**

(1) *Update a basis* $\mathbf{S} \in \mathbb{R}^{n\times m}$ *for the subspace* $\mathcal{S}$ *of dimension* $m < n$.
(2) *Perform a Rayleigh-Ritz analysis:*
Solve the projected eigenvalue problem $\mathbf{S}^T\mathbf{K}\mathbf{S}\mathbf{y} = \mathbf{S}^T\mathbf{M}\mathbf{S}\mathbf{y}\theta$.
(3) *Form the residual:* $\mathbf{r} = \mathbf{K}\mathbf{x} - \mathbf{M}\mathbf{x}\theta$, *where* $\mathbf{x} = \mathbf{S}\mathbf{y}$ *is called a Ritz vector and* $\theta = \rho(\mathbf{x})$ *a Ritz value.*
(4) *Flag a Ritz pair* $(\mathbf{x}, \rho(\mathbf{x}))$ *if the corresponding residual satisfies the specified convergence criterion.*

referred to as the *outer loop* or iteration because Algorithm 2.1 is typically one step of an iteration where the step (1) can invoke a further *inner* iteration.

2.2. The subspace $\mathcal{S}$

A distinguishing characteristic of all the algorithms is the size, or number of basis vectors, m of the subspace $\mathcal{S}$ when a Rayleigh-Ritz analysis is performed. The size of the basis $\mathbf{S}$ is either

constant or increases. Examples of the former are the gradient-based methods DACG [18, 5, 6] and LOBPCG [25] while examples of the latter are the Davidson algorithm [12], the Jacobi-Davidson algorithm [33], and the shift-invert Lanczos algorithm [13].

After step (4) of Algorithm 2.1, the subspace $\mathcal{S}$ is updated. Different choices are possible.

- Exploiting the property that a stationary point of the Rayleigh quotient is a zero of the gradient
$$\mathbf{g}(\mathbf{x}) = \mathbf{grad}\ \rho(\mathbf{x}) = \frac{2}{\mathbf{x}^T\mathbf{M}\mathbf{x}}(\mathbf{K}\mathbf{x} - \mathbf{M}\mathbf{x}\rho(\mathbf{x})) = \frac{2}{\mathbf{x}^T\mathbf{M}\mathbf{x}}\mathbf{r}(\mathbf{x}),$$
where $\mathbf{r}(\mathbf{x})$ is defined to be the residual and is proportional to the gradient when $\mathbf{x}$ is properly normalized, a Newton step gives the correction vector
$$\mathbf{t} = -\left(\frac{\partial \mathbf{g}}{\partial \mathbf{x}}(\mathbf{x})\right)^{-1}\mathbf{g}(\mathbf{x}). \tag{4}$$
If we require that $\mathbf{x}^T\mathbf{M}\mathbf{t} = 0$, then equation (4) is mathematically equivalent to
$$\begin{bmatrix}\mathbf{K} - \rho(\mathbf{x})\mathbf{M} & \mathbf{M}\mathbf{x} \\ \mathbf{x}^T\mathbf{M} & 0\end{bmatrix}\begin{pmatrix}\mathbf{t} \\ \mu\end{pmatrix} = -\frac{2}{\mathbf{x}^T\mathbf{M}\mathbf{x}}\begin{pmatrix}\mathbf{r}(\mathbf{x}) \\ 0\end{pmatrix}, \tag{5}$$
where μ is a Lagrange multiplier enforcing the $\mathbf{M}$-orthogonality of $\mathbf{t}$ against $\mathbf{x}$. The Rayleigh quotient iteration uses the exact solution $\mathbf{t}$, while the Davidson and Jacobi-Davidson algorithms approximate this correction vector. We refer the reader to the papers [33, 14, 39, 42] and the references therein for further details.
- The update is the search direction of a gradient method applied to the minimization of the Rayleigh-quotient (3). Hestenes and Karush [22] proposed an algorithm *à la* steepest descent, where the update direction $\mathbf{p}$ is proportional to the gradient $\mathbf{g}(\mathbf{x})$. Bradbury and Fletcher [9] introduced a conjugate gradient-type algorithm where consecutive update directions $\mathbf{p}$ are $\mathbf{K}$-orthogonal. Knyazev [25] employs a three-term recurrence. We refer the reader to the papers [15, 29, 17, 4, 32, 24] for variants including the incorporation of preconditioning, convergence analysis, and deflation schemes.
- The update $\mathbf{z}$ is the solution of the linear set of equations
$$(\mathbf{K} - \sigma\mathbf{M})\mathbf{z} = \mathbf{M}\mathbf{q}_k, \tag{6}$$
where the subspace $\{\mathbf{q}_0, \cdots, \mathbf{q}_k, \mathbf{z}\}$ defines a Krylov subspace for $(\mathbf{K} - \sigma\mathbf{M})^{-1}\mathbf{M}$ generated by the starting vector $\mathbf{q}_0$. The reader is referred to [13, 34, 20] for further information.

Preconditioning can be incorporated in all the updates. The gradient-based methods are accelerated by applying the preconditioner $\mathbf{N}$ to the gradient $\mathbf{g}$ (or the residual $\mathbf{r}$) while the Newton-based schemes and the shift-invert Lanczos method employ preconditioned iterative methods for the solution of the associated sets of linear equations.

Finally, except for the ARPACK implementation of the Lanczos algorithm, our algorithms incorporate an explicit deflation (or locking) step when a Ritz vector satisfies the convergence criterion. In our implementations, the columns of $\mathbf{S}$ satisfy
$$\mathbf{Q}^T\mathbf{M}\mathbf{S} = \mathbf{0},$$
where $\mathbf{Q}$ contains the converged Ritz vectors.

2.3. Some notations

In the remainder of the paper, we provide pseudocode for the algorithms we employed in our comparison. The notation $\alpha := \beta$ denotes that α is overwritten with the results of β. The following functions will be used repeatedly within the pseudocodes provided.

1. $(\mathbf{Y}, \mathbf{\Theta}) := \mathrm{RR}(\mathbf{S}, b)$ performs a Rayleigh-Ritz analysis where eigenvectors $\mathbf{Y}$ and eigenvalues $\mathbf{\Theta}$ are computed for the pencil $(\mathbf{S}^T\mathbf{K}\mathbf{S}, \mathbf{S}^T\mathbf{M}\mathbf{S})$. The first b pairs with smallest Ritz values are returned in $\mathbf{Y}$ and $\mathbf{\Theta}$ in a nondecreasing order.
2. $\mathbf{Y} := \mathrm{ORTHO}(\mathbf{X}, \mathbf{Q})$ denotes that $\mathbf{Y} = (\mathbf{I} - \mathbf{Q}\mathbf{Q}^T\mathbf{M})\mathbf{X}$ where $\mathbf{Q}^T\mathbf{M}\mathbf{Q} = \mathbf{I}$.
3. $\mathbf{Y} := \mathrm{QR}(\mathbf{X})$ denotes that the output matrix $\mathbf{Y}$ satisfies $\mathbf{Y}^T\mathbf{M}\mathbf{Y} = \mathbf{I}$ and $\mathrm{Range}(\mathbf{Y}) = \mathrm{Range}(\mathbf{X})$.
4. $\mathrm{size}(\mathbf{X}, 1)$ and $\mathrm{size}(\mathbf{X}, 2)$ denote the number of rows and columns of $\mathbf{X}$, respectively.
5. The matrix $\mathbf{Q}$ always denotes the Ritz vectors that satisfy the convergence criterion; $\mathbf{K}, \mathbf{M}$, and $\mathbf{N}$ denote the stiffness, mass, and preconditioning matrices. Application of $\mathbf{N}$ to a vector $\mathbf{b}$ implies computing the vector $\mathbf{N}^{-1}\mathbf{b}$.

The generic function $\mathrm{RR}(\cdot, \cdot)$ invokes the appropriate LAPACK [2] subroutine. The generic functions $\mathrm{ORTHO}(\cdot)$ and $\mathrm{QR}(\cdot)$ implement a classical block Gram-Schmidt algorithm [3, p. 186] with iterative refinement [23, 8].

3. Schemes with *constant*-size subspaces

This Section describes two schemes for minimizing the Rayleigh quotient on a subspace with a fixed size. The first scheme is based on the Bradbury and Fletcher [9] conjugate gradient algorithm. The second scheme uses a three-term recurrence.

3.1. The block deflation-accelerated conjugate gradient (BDACG) Algorithm

Algorithm 3.2 lists the iteration associated with BDACG. We have adopted this name from a series of papers [18, 5, 6, 7] that present an unblocked scheme. The iteration is continued until the desired number of eigenpairs are approximated.

The space for the minimization of the Rayleigh quotient is the span of $[\mathbf{X}_k, \mathbf{P}_k]$. The block of vectors $\mathbf{P}_k$ is our block extension for the search direction introduced by the gradient scheme of Bradbury-Fletcher [9]. BDACG-(5), i.e. step (5) in the BDACG Algorithm 3.2, computes the search directions with the preconditioned residuals instead of the preconditioned gradients because the columns of $\mathbf{X}_k$ are $\mathbf{M}$-orthonormal. We remark that each column vector of $\mathbf{P}_k$ is only $\mathbf{K}$-orthogonal to the corresponding column vector of $\mathbf{P}_{k-1}$. Enforcing the stronger condition $\mathbf{P}_k^T\mathbf{K}\mathbf{P}_{k-1} = \mathbf{0}$ lead to a more expensive iteration with no reduction in the number of (outer) iterations.

BDACG-(6) $\mathbf{M}$-orthogonalizes the current search directions $\mathbf{P}_k$ against the column span of $\mathbf{Q}$ that contains Ritz vectors that have been deflated. Our experiments revealed that this orthogonalization prevented copies of Ritz values from emerging during the course of the iteration. In our experiments, we determined that the eigenvectors computed in BDACG-(7) needed to be scaled so that the diagonal elements of $\mathbf{Y}_k$ are nonnegative. This is a generalization of quadratic line search that retains the positive root in the unblocked algorithm.

ALGORITHM 3.2: **(BDACG) Block deflation-accelerated conjugate gradient Algorithm**

(1) *Select a random* $\tilde{\mathbf{X}}_0 \in \mathbb{R}^{n\times b}$ *where* $1 \le b < n$ *is the blocksize;* $\mathbf{X}_0 := \tilde{\mathbf{X}}_0\tilde{\mathbf{Y}}_0$ *where* $(\tilde{\mathbf{Y}}_0, \Theta_0) := \mathrm{RR}(\tilde{\mathbf{X}}_0, b)$ *and let* $\mathbf{R}_0 := \mathbf{K}\mathbf{X}_0 - \mathbf{M}\mathbf{X}_0\Theta_0$.
(2) *Set* $k := 0$ *and* $\mathbf{Q} := []$.
(3) Until $\mathrm{size}(\mathbf{Q}, 2) \ge nev$ do
(4) *Solve the preconditioned linear system* $\mathbf{N}\mathbf{H}_k = \mathbf{R}_k$.
(5) If $k = 0$ then
$\mathbf{P}_k := -\mathbf{H}_k$ *and* $\mathbf{B}_k := \mathrm{diag}(\mathbf{H}_k^T\mathbf{R}_k)$.
else
$\mathbf{P}_k := -\mathbf{H}_k + \mathbf{P}_{k-1}\mathbf{B}_k$ *and* $\mathbf{B}_k := \mathrm{diag}(\mathbf{H}_k^T\mathbf{R}_k)\mathbf{B}_{k-1}^{-1}$.
end if.
(6) $\mathbf{P}_k := \mathrm{ORTHO}(\mathbf{P}_k, \mathbf{Q})$.
(7) *Let* $\mathbf{S}_k := [\mathbf{X}_k, \mathbf{P}_k]$ *and compute* $(\mathbf{Y}_k, \Theta_{k+1}) := \mathrm{RR}(\mathbf{S}_k, b)$.
(8) $\mathbf{X}_{k+1} := \mathbf{S}_k\mathbf{Y}_k$.
(9) $\mathbf{R}_{k+1} := \mathbf{K}\mathbf{X}_{k+1} - \mathbf{M}\mathbf{X}_{k+1}\Theta_{k+1}$.
(10) $k := k + 1$.
(11) If *some columns of* $\mathbf{R}_k$ *satisfy the convergence criterion* then
Augment $\mathbf{Q}$ *with the corresponding Ritz vectors from* $\mathbf{X}_k$*;*
set $k := 0$ *and define new* $\mathbf{X}_0$ *and* $\mathbf{R}_0$.
end if.
(12) end Until.

BDACG-(11) deflates Ritz vectors from $\mathbf{X}_k$ when they satisfy the convergence criterion. The new columns of $\mathbf{X}_0$ are defined by the vectors not-deflated and the vectors associated with the next largest Ritz values. In practice, the generic function $\mathrm{RR}(\cdot,\cdot)$ calls the LAPACK routine DSYGV that computes the $2b$ eigenpairs of the projected eigenproblem.

The matrices $\mathbf{K}$, $\mathbf{M}$, and $\mathbf{N}$ are accessed only once per iteration (except where $\mathbf{M}$ is used for deflation). Therefore, we store the block vectors $\mathbf{X}_k$, $\mathbf{K}\mathbf{X}_k$, $\mathbf{M}\mathbf{X}_k$, $\mathbf{P}_k$, $\mathbf{K}\mathbf{P}_k$, $\mathbf{M}\mathbf{P}_k$, $\mathbf{R}_k$, and $\mathbf{H}_k$. When the first nev eigenpairs are requested, the overall storage requirements for BDACG are:

- a vector of length nev elements (for the converged Ritz values),
- nev vectors of length n (for the converged Ritz vectors),
- $8 \cdot b$ vectors of length n,
- $O(b^2)$ elements (for the Rayleigh-Ritz analysis).

In our experiments, the matrix $\mathbf{B}_k$ remained non-singular throughout the computation and the Rayleigh-Ritz analysis never failed. However, for the sake of robustness, we have equipped BDACG with a restart when one of these failures occurs.

ALGORITHM 3.3: **(LOBPCG) Locally-optimal block preconditioned conjugate gradient method**

(1) *Select a random* $\tilde{\mathbf{X}}_0 \in \mathbb{R}^{n\times b}$ *where* $1 \leq b < n$ *is the blocksize;* $\mathbf{X}_0 := \tilde{\mathbf{X}}_0\tilde{\mathbf{Y}}_0$ *where* $(\tilde{\mathbf{Y}}_0, \Theta_0) := \mathrm{RR}(\tilde{\mathbf{X}}_0, b)$ *and let* $\mathbf{R}_0 := \mathbf{K}\mathbf{X}_0 - \mathbf{M}\mathbf{X}_0\Theta_0$.
(2) *Set* $k := 0$, $\mathbf{Q} := []$, *and* $\mathbf{P}_0 := []$.
(3) Until size$(\mathbf{Q}, 2) \geq nev$ do
(4) *Solve the preconditioned linear system* $\mathbf{N}\mathbf{H}_k = \mathbf{R}_k$.
(5) $\mathbf{H}_k := \mathrm{ORTHO}(\mathbf{H}_k, \mathbf{Q})$.
(6) *Let* $\mathbf{S}_k := [\mathbf{X}_k, \mathbf{H}_k, \mathbf{P}_k]$ *and compute* $(\mathbf{Y}_k, \Theta_{k+1}) := \mathrm{RR}(\mathbf{S}_k, b)$.
(7) $\mathbf{X}_{k+1} := [\mathbf{X}_k, \mathbf{H}_k, \mathbf{P}_k]\mathbf{Y}_k$.
(8) $\mathbf{P}_{k+1} := [\mathbf{0}, \mathbf{H}_k, \mathbf{P}_k]\mathbf{Y}_k$.
(9) $\mathbf{R}_{k+1} := \mathbf{K}\mathbf{X}_{k+1} - \mathbf{M}\mathbf{X}_{k+1}\Theta_{k+1}$.
(10) $k := k + 1$.
(11) If *some columns of* $\mathbf{R}_k$ *satisfy the convergence criterion* then
Augment $\mathbf{Q}$ *with the corresponding Ritz vectors from* $\mathbf{X}_k$*;*
set $k := 0$ *and define new* $\mathbf{X}_0$ *and* $\mathbf{R}_0$.
end if.
(12) end Until.

3.2. The locally-optimal block preconditioned conjugate gradient (LOBPCG) Algorithm

In contrast to BDACG, Knyazev [25] suggests that the space for the minimization be augmented by the span of $\mathbf{H}_k$. The resulting algorithm is deemed *locally-optimal* because the Rayleigh quotient ρ is minimized with respect to all available vectors. Mathematically, the span of $[\mathbf{X}_k, \mathbf{H}_k, \mathbf{P}_k]$ is equal to the span of $[\mathbf{X}_k, \mathbf{H}_k, \mathbf{X}_{k-1}]$. The columns of the former matrix are better conditioned than the columns of the latter matrix.

Algorithm 3.3 lists the iteration associated with LOBPCG. Our implementation of LOBPCG differs from the one presented in [25]. For instance, we employ explicit deflation and allow the block size b to be independent of the number of Ritz pairs desired. The iteration is continued until the desired number of eigenpairs are approximated.

LOBPCG-(5) $\mathbf{M}$-orthogonalizes the current preconditioned residuals $\mathbf{H}_k$ against column span of $\mathbf{Q}$ that contains Ritz vectors that have been deflated. Our experiments revealed that this orthogonalization prevented copies of Ritz values from emerging during the course of the iteration.

LOBPCG-(11), as in BDACG, deflates Ritz vectors from $\mathbf{X}_k$ when they satisfy the convergence criterion. The new columns of $\mathbf{X}_0$ are defined by the vectors not-deflated and the vectors associated with the next largest Ritz values. In practice, the generic function $\mathrm{RR}(\cdot,\cdot)$ calls the LAPACK routine DSYGV.

The matrices $\mathbf{K}$, $\mathbf{M}$, and $\mathbf{N}$ are accessed only once per iteration (except where $\mathbf{M}$ is used for deflation). Therefore, we store the block vectors $\mathbf{X}_k$, $\mathbf{K}\mathbf{X}_k$, $\mathbf{M}\mathbf{X}_k$, $\mathbf{H}_k$, $\mathbf{K}\mathbf{H}_k$, $\mathbf{M}\mathbf{H}_k$, $\mathbf{P}_k$, $\mathbf{K}\mathbf{P}_k$, $\mathbf{M}\mathbf{P}_k$, and $\mathbf{R}_k$. When the first *nev* eigenpairs are requested, the overall storage requirements for the algorithm LOBPCG are:

- a vector of length *nev* elements (for the converged Ritz values),
- *nev* vectors of length n (for the converged Ritz vectors),
- $10 \cdot b$ vectors of length n,
- $O(b^2)$ elements (for the Rayleigh-Ritz analysis).

In our experiments, the Rayleigh-Ritz analysis never failed. But, for the sake of robustness, we have equipped our code with a restart when the routine DSYGV fails.

4. Schemes with subspaces that *increase* in size

In this Section, we present three schemes for minimizing the Rayleigh quotient on a subspace with a varying size. The first two are Newton-based schemes and the third is a shift-invert Lanczos method.

4.1. The block Davidson Algorithm

ALGORITHM 4.4: **Block Davidson Algorithm**

(1) *Select a random* $\tilde{\mathbf{X}}_0 \in \mathbb{R}^{n\times b}$ *where* $1 \le b < n$ *is the blocksize;* $\mathbf{X}_0 := \tilde{\mathbf{X}}_0\tilde{\mathbf{Y}}_0$ *where* $(\tilde{\mathbf{Y}}_0, \Theta_0) := \mathrm{RR}(\tilde{\mathbf{X}}_0, b)$ *and let* $\mathbf{R}_0 := \mathbf{K}\mathbf{X}_0 - \mathbf{M}\mathbf{X}_0\Theta_0$.
(2) *Set* $k := 0$, $\mathbf{Q} := []$, *and* $\mathbf{S}_0 := [\mathbf{X}_0]$.
(3) Until $\mathrm{size}(\mathbf{Q}, 2) \ge nev$ do
(4) *Solve the preconditioned linear system* $\mathbf{N}\mathbf{H}_k = \mathbf{R}_k$.
(5) $\mathbf{H}_k := \mathrm{ORTHO}(\mathbf{H}_k, [\mathbf{Q}, \mathbf{S}_k])$.
(6) $\mathbf{H}_k := \mathrm{QR}(\mathbf{H}_k)$.
(7) *Let* $\mathbf{S}_{k+1} := [\mathbf{S}_k, \mathbf{H}_k]$ *and compute* $(\mathbf{Y}_{k+1}, \Theta_{k+1}) := \mathrm{RR}(\mathbf{S}_{k+1}, b)$.
(8) $\mathbf{X}_{k+1} := \mathbf{S}_{k+1}\mathbf{Y}_{k+1}$.
(9) $\mathbf{R}_{k+1} := \mathbf{K}\mathbf{X}_{k+1} - \mathbf{M}\mathbf{X}_{k+1}\Theta_{k+1}$.
(10) $k := k + 1$.
(11) If *any columns of* $\mathbf{R}_k$ *satisfy the convergence criterion* then
Augment $\mathbf{Q}$ *with the corresponding Ritz vectors from* $\mathbf{X}_k$ *and restart to obtain an updated* $\mathbf{S}_k$.
end if.
(12) If *the dimensions of* $\mathbf{S}_k$ *reach the limit of storage allocated* then
Restart to obtain an updated $\mathbf{S}_k$.
end if.
(13) end Until.

Algorithm 4.4 lists the iteration associated with our block extension of the Davidson algorithm (see [31, 36] for alternate block variants and references). The iteration is continued until the desired number of eigenpairs are approximated.

At the k-th iteration, the subspace $\mathcal{S}_k$ for the minimization of the Rayleigh quotient is spanned by the $\mathbf{M}$-orthonormal basis $\mathbf{S}_k$. To enrich the subspace at each iteration, we use

the preconditioned residuals $\mathbf{N}^{-1}\mathbf{R}_k$ as an approximation to the correction vector $\mathbf{T}$ for the Newton step of equation (4). This approximation differs from the Davidson algorithm [12] in that we employ a fixed preconditioner $\mathbf{N}$ at every step.

The step Davidson-(5) $\mathbf{M}$-orthogonalizes the columns of $\mathbf{H}_k$ against the deflated Ritz vectors stored in $\mathbf{Q}$ and the basis $\mathbf{S}_k$. Then Davidson-(6) $\mathbf{M}$-orthonormalizes the resulting vectors.

The generic function $\mathrm{RR}(\cdot,\cdot)$ in Davidson-(7) calls the LAPACK routine DSYEV that computes all the eigenpairs of the projected eigenproblem.

Davidson-(11) deflates Ritz vectors from $\mathbf{X}_k$ when they satisfy the convergence criterion. The columns of the residual matrix $\mathbf{R}_k$ associated with deflated Ritz vectors are replaced with the Ritz vectors corresponding to the next largest Ritz values.

Davidson-(12) limits the dimension of the subspace basis $\mathbf{S}_k$. When *nev* eigenpairs are requested, the number of vectors allocated for the storage of $[\mathbf{Q}, \mathbf{S}_k]$ is $2 \cdot nev + b$ and this combined storage represents the working subspace. As the iteration progresses and Ritz pairs converge, the number of vectors in the active subspace $\mathbf{S}_k$ is bounded by

$$2 \cdot nev - \lfloor \frac{\mathrm{size}(\mathbf{Q}, 2)}{b} \rfloor \cdot b, \tag{7}$$

where $\lfloor \cdot \rfloor$ is the floor function.

Both Davidson-(11) and Davidson-(12) effect a restart. Suppose that the number of columns of $\mathbf{S}_k$ is $p \cdot b$. Then we restart by multiplying $\mathbf{S}_k$ with the $\lfloor p/2 \rfloor \cdot b$ columns of $\mathbf{Y}_k$ associated with the smallest Ritz values not deflated. The choice of $\lfloor p/2 \rfloor \cdot b$ columns for the restart of $\mathbf{S}_k$ is a balance between a number large enough so that the loss of information is minimized and small enough so that a useful subspace $\mathbf{S}_k$ is constructed before the limit (7) on the number of columns is attained. Restarting with Ritz vectors associated with the smallest Ritz values worked better in practice than using random vectors. The paper [37] discusses related restart strategies.

The preconditioner $\mathbf{N}$ is applied only once per iteration, while the matrices $\mathbf{K}$ and $\mathbf{M}$ are accessed twice per iteration (except where $\mathbf{M}$ is used for deflation and orthonormalization). When the first *nev* eigenpairs are requested and with the upper limit (7), the overall storage requirements for our implementation of Davidson are:

- a vector of length *nev* words (for the converged Ritz values),
- $2 \cdot nev$ vectors of length n (the converged Ritz vectors are stored in the initial *nev* vectors),
- $4 \cdot b$ vectors of length n,
- $O(nev^2)$ elements (for the Rayleigh-Ritz analysis),
- $O(b^2)$ elements.

Our implementation proved stable during all our experiments.

4.2. The block Jacobi-Davidson conjugate gradient Algorithm (BJDCG)

Our second scheme for minimizing the Rayleigh quotient by expanding the subspace is a block extension of a Jacobi–Davidson algorithm [33]. We implement a block version of the Jacobi–Davidson variant due to Notay [28] tailored for symmetric matrices. We do not list an algorithm for BJDCG because the differences with Algorithm 4.4 are slight. The algorithms differ mainly in the enrichment of the subspace. Step (4) of Algorithm 4.4 is replaced by

$$\mathbf{H}_k = \mathrm{CORRECTION}(\mathbf{R}_k, \mathbf{N}, \hat{\mathbf{Q}}, \tau, \epsilon), \qquad \text{where} \quad \hat{\mathbf{Q}} := [\mathbf{Q}, \mathbf{X}_k],$$

that solves the correction equation

$$(\mathbf{I}-\mathbf{M}\hat{\mathbf{Q}}\hat{\mathbf{Q}}^T)(\mathbf{K}-\tau\mathbf{M})(\mathbf{I}-\hat{\mathbf{Q}}\hat{\mathbf{Q}}^T\mathbf{M})\mathbf{T}=-\mathbf{R}_k \quad \text{with } \hat{\mathbf{Q}}^T\mathbf{M}\mathbf{T}=\mathbf{0} \tag{8}$$

with a block preconditioned conjugate gradient (BPCG) algorithm. The correction equation (8) is equivalent to the block extension of equation (5) with righthand side

$$-\begin{pmatrix}\mathbf{R}_k\\ \mathbf{0}\end{pmatrix}.$$

Algorithm 4.5 lists the BPCG iteration we used to solve the correction equation (8). Step

ALGORITHM 4.5: **Routine** $\mathbf{H}=\text{CORRECTION}(\mathbf{R},\mathbf{N},\hat{\mathbf{Q}}=[\mathbf{Q},\mathbf{X}],\tau,\epsilon)$

(1) *Set* $j:=0$, $\mathbf{T}_0:=\mathbf{0}$, *and* $\tilde{\mathbf{R}}_0:=-\mathbf{R}$.
(2) While $j<1{,}000$ do
(3) $\quad\mathbf{W}_j:=\left[\mathbf{I}-\mathbf{N}^{-1}\mathbf{M}\hat{\mathbf{Q}}\left(\hat{\mathbf{Q}}^T\mathbf{M}\mathbf{N}^{-1}\mathbf{M}\hat{\mathbf{Q}}\right)^{-1}\hat{\mathbf{Q}}^T\mathbf{M}\right]\mathbf{N}^{-1}\tilde{\mathbf{R}}_j.$
(4) $\quad$If $j=0$ then
$\quad\quad\mathbf{P}_0:=\mathbf{W}_0.$
$\quad$else
$\quad\quad\mathbf{P}_j:=\left[\mathbf{I}-\mathbf{P}_{j-1}\left(\mathbf{P}_{j-1}^T(\mathbf{K}-\tau\mathbf{M})\mathbf{P}_{j-1}\right)^{-1}\mathbf{P}_{j-1}^T(\mathbf{K}-\tau\mathbf{M})\right]\mathbf{W}_j.$
$\quad$end if.
(5) $\quad\mathbf{T}_{j+1}:=\mathbf{T}_j+\mathbf{P}_j\left(\mathbf{P}_j^T(\mathbf{K}-\tau\mathbf{M})\mathbf{P}_j\right)^{-1}\mathbf{P}_j^T\tilde{\mathbf{R}}_j.$
(6) $\quad\tilde{\mathbf{R}}_{j+1}:=\tilde{\mathbf{R}}_j-(\mathbf{K}-\tau\mathbf{M})\mathbf{P}_j\left(\mathbf{P}_j^T(\mathbf{K}-\tau\mathbf{M})\mathbf{P}_j\right)^{-1}\mathbf{P}_j^T\tilde{\mathbf{R}}_j.$
(7) $\quad j:=j+1.$
(8) $\quad$If *all the columns of* $\tilde{\mathbf{R}}_j$ *satisfy the convergence criterion* then
$\quad\quad$*Exit the loop.*
$\quad$end if
(9) $\quad$If $j>1$ then
$\quad\quad$*Check the eigenresiduals associated with* $\mathbf{X}+\mathbf{T}_j$ *for an early exit.*
$\quad$end if.
(10) end While.
(11) $\mathbf{H}:=\mathbf{T}_j.$

(3) applies the preconditioner $\mathbf{N}$ to the correction equation (8) (see Geus' thesis [19] for details). Step (4) computes the block of search directions, and steps (5)–(6) compute the j-th approximation to the correction equation and corresponding residual, respectively. Step (8) terminates the BPCG iteration when the Euclidean norms of the columns of $\tilde{\mathbf{R}}_j$ have been reduced by a factor of ϵ relative to the columns of $\tilde{\mathbf{R}}_0$. The tolerance ϵ used in Algorithm 4.5 is set equal to $2^{-\ell}$ where ℓ is a counter on the number of (outer) BJDCG iterations needed to compute a Ritz value (see [39, p.130] for a discussion).

The coefficient τ is set to 0 when the norm of residuals in $\mathbf{R}_k$ are larger than a given tolerance. Otherwise, τ is set to the smallest Ritz value in Θ_k. The reader is referred to [16] for further details.

In contrast to step (8) that checks for termination of the BPCG algorithm, step (9) checks the Ritz pair residuals. Because $\mathbf{T}_j$ is the approximation to the correction equation, we define

$$\mathbf{V} = (\mathbf{X} + \mathbf{T}_j)\mathbf{Y}, \quad \text{where } (\mathbf{Y}, \Theta) = \mathrm{RR}(\mathbf{X} + \mathbf{T}_j, b),$$

and check the columns norms of $\mathbf{KV} - \mathbf{MV}\Theta$. If any column norm stagnates, increases in norm, or satisfies the convergence criterion, then we exit the BPCG iteration with the current approximation $\mathbf{T}_j$ (see Notay's paper [28] for further details and discussion).

A generalization of the proof given by Notay [28] to the generalized symmetric positive definite eigenvalue problem shows that $\mathbf{P}_j^T(\mathbf{K} - \tau\mathbf{M})\mathbf{P}_j$ is symmetric positive definite (on the space orthogonal to the range of $\hat{\mathbf{Q}}$). For robustness, if this matrix becomes indefinite, we exit the BPCG loop and perform a restart of the search space $\mathbf{S}_k$ in BJDCG. In our numerical experiments, the maximum number of BPCG iterations was never reached.

When the first *nev* eigenpairs are requested, the number of column vectors allocated for the storage of $[\mathbf{Q}, \mathbf{S}_k]$ is also $2 \cdot nev + b$. Therefore, the overall storage requirements for BJDCG are:

- a vector of *nev* elements (for the converged Ritz values),
- $2 \cdot nev$ vectors of length n (the converged Ritz vectors are stored in the initial *nev* vectors),
- *nev* vectors of length n (for storing $\mathbf{N}^{-1}\mathbf{MQ}$),
- $5 \cdot b$ vectors of length n,
- $4 \cdot b$ vectors of length n (for the block PCG),
- $O(nev^2)$ elements (for the Rayleigh-Ritz analysis),
- $O(b^2)$ elements.

The storage requirements of BJDCG are the largest of all the algorithms we compared.

4.3. The shift-invert Lanczos Algorithm

Algorithm 4.6 lists the iteration associated with the shift-invert Lanczos algorithm, when the *nev* eigenpairs closest to σ are requested. Our implementation of Algorithm 4.6 is based on the implicitly restarted Lanczos method in ARPACK. For detailed comments, we refer the reader to the users' guide [27]. In our experiments, we are interested in the smallest eigenvalues of the generalized eigenvalue (1). Therefore, we set the shift σ to 0.

At the k-th iteration, the subspace $\mathcal{S}_k$ for the minimization of the Rayleigh quotient is spanned by the $\mathbf{M}$-orthonormal basis $\mathbf{S}_k$. Lanczos-(4) defines the new direction for enriching the subspace $\mathcal{S}_k$ at each iteration. We use an AMG-preconditioned conjugate gradient iteration as an inner iteration to solve the linear system.

Lanczos-(5) and Lanczos-(6) $\mathbf{M}$-orthogonalize the new direction $\mathbf{z}$ against the basis $\mathbf{S}_k$ and $\mathbf{M}$-orthonormalize the resulting vector, respectively.

Lanczos-(9) limits the dimension of the subspace basis $\mathbf{S}_k$. When *nev* eigenpairs are requested, the number of column vectors allocated for the storage of $\mathbf{S}_k$ is $2 \cdot$ *nev*.

In step Lanczos-(9a), the projected eigenproblem is tridiagonal and automatically generated by the Lanczos iteration and so an explicit projection with $\mathbf{S}_k$ in the Rayleigh-Ritz analysis is not needed. The generic function $\mathrm{RR}(\cdot, \cdot)$ calls the ARPACK routine DSEIGT (based on a modification of the LAPACK routine DSTQR) to compute all the eigenpairs of the tridiagonal matrix. We remark that in contrast to other eigensolvers, shift-invert Lanczos requires the largest *nev* eigenvalues of the projection matrix.

ALGORITHM 4.6: **Shift-invert Lanczos Algorithm**

(1) *Select a random* $\tilde{\mathbf{q}}_0 \in \mathbb{R}^n$; $\mathbf{q}_0 := \mathrm{QR}(\tilde{\mathbf{q}}_0)$.
(2) *Set* $k := 0$, *nconv* $:= 0$, *and* $\mathbf{S}_0 := [\mathbf{q}_0]$.
(3) Until *nconv* $\geq$ *nev* do
(4) *Solve* $(\mathbf{K} - \sigma\mathbf{M})\mathbf{z} = \mathbf{M}\mathbf{q}_k$.
(5) $\mathbf{z} := \mathrm{ORTHO}(\mathbf{z}, \mathbf{S}_k)$.
(6) $\mathbf{q}_{k+1} := \mathrm{QR}(\mathbf{z})$.
(7) $\mathbf{S}_{k+1} := [\mathbf{S}_k, \mathbf{q}_{k+1}]$.
(8) $k := k + 1$.
(9) If *the dimensions of* $\mathbf{S}_k$ *reach the limit of storage allocated* then
(9a) *Compute* $(\mathbf{Y}_k, \mathbf{\Theta}_k) := \mathrm{RR}(\mathbf{S}_k, nev)$.
(9b) $\mathbf{R}_k := \mathbf{K}\mathbf{S}_k\mathbf{Y}_k - \mathbf{M}\mathbf{S}_k\mathbf{Y}_k(\sigma\mathbf{I} + \mathbf{\Theta}_k^{-1})$.
(9c) Let nconv denote the number of Ritz pairs that satisfy the convergence criterion; Exit the outer loop if nconv $\geq$ *nev.*
(9d) *Restart to obtain an updated* $\mathbf{S}_k$.
end if.
(10) end Until.

Steps Lanczos-(9b) and Lanczos-(9c) monitor the convergence of the Ritz pairs by explicitly computing the first *nev* residuals $\mathbf{R}_k$ and testing each column against our convergence criterion. This is in contrast to the ARPACK convergence check [27] that monitors the convergence of the eigenpairs of the shift-invert system via Ritz estimates. The explicit computation of the residuals required us to edit the ARPACK source code to include a reverse communication step so as to allow the code calling ARPACK to compute the residuals.

Finally, because $\mathbf{S}_k$ holds a maximum of $2\cdot$ *nev* vectors, Lanczos-(9d) implements implicit restarting. We refer the reader to [27] for specific details on implicit restarting but in analogy with the block Davidson algorithm, $\mathbf{S}_k$ is compressed into a matrix with less columns containing the best approximation to the smallest eigenvalues. The number of columns after restarting is

$$nev + \max(nconv, 0.5 \cdot nev)$$

where *nconv* denotes the number of Ritz pairs that satisfy the convergence criterion. The reader is referred to [27] for further details. Increasing the number of columns by the number of converged Ritz pairs effects an *implicit* deflation or equivalently *soft-locking* [25] mechanism.

In our experiments, when the smallest *nev* eigenpairs are requested, the overall storage requirements for shift-invert Lanczos are:

- a vector of length *nev* elements (for the converged Ritz values),
- $2 \cdot$ *nev* vectors of length n (for storing $\mathbf{S}_k$ and the converged Ritz vectors),
- 5 vectors of length n,
- $O(nev^2)$ elements,
- 3 vectors of length n (for conjugate gradient algorithm),
- $6\min(nev, 5)$ vectors of length n (for computing the residuals).

We remark, that unlike the previous algorithms, ARPACK does not check for convergence at each outer iteration. Instead, convergence of Ritz pairs is determined at restart. Moreover, there is no explicit deflation step only a postprocessing step to overwrite the Lanczos vectors with Ritz vectors upon convergence of *nev* (or more) Ritz pairs.

5. Numerical experiments

In this Section, we discuss the numerical experiments used for the comparisons. The codes are implemented in C++, using the Trilinos [21] project. This project provides, through a collection of classes, the algebraic operations, the smoothed aggregation AMG preconditioner, and the preconditioned conjugate gradient algorithm. For the shift-invert Lanczos algorithm, our C++ code invokes the Fortran 77 package ARPACK [27].

Inside Trilinos, the linear algebra class, namely Epetra, manipulates the vectors, the blocks of vectors (or multivectors), and the sparse matrices. All these objects are distributed across the processors. Whenever possible, Epetra implements the algebraic operations block-wise. For instance, the matrices $\mathbf{K}$ and $\mathbf{M}$ can be applied efficiently to a block of vectors.

Because ARPACK is a publicly available high-quality Lanczos implementation that includes a distributed memory implementation, we present the normalized timings

$$\frac{\text{time for an eigensolver}}{\text{time for ARPACK}} \tag{9}$$

where eigensolver is in turn BDACG, LOBPCG, block Davidson, and BJDCG. The initial vectors used by all the algorithms are generated using a random number generator. In addition to reporting the size of residuals, all the algorithms checked the orthonormality of the Ritz vectors computed via the check

$$\max_{i,j=1,\ldots,\text{nev}} |\mathbf{e}_i^T(\mathbf{X}^T\mathbf{M}\mathbf{X} - \mathbf{I})\mathbf{e}_j|. \tag{10}$$

We first describe some details associated with the AMG preconditioner in subsection 5.1 followed by the results of our experiments on three problems.

5.1. AMG Preconditioner

The package ML provides the smoothed aggregation AMG preconditioner [41]. Several recent studies have demonstrated the viability of AMG preconditioners for problems in computational mechanics [1, 38]. In addition to the public domain package ML, the commercial finite element analysis program ANSYS now provides an AMG-based solver [30].

The smoothed aggregation AMG algorithm requires no geometric information and therefore is attractive for complex domains with unstructured meshes. The basic idea of all AMG algorithms is to capture errors by utilizing multiple resolutions. High energy (or oscillatory) components are effectively reduced through a simple smoothing procedure, while low energy (or smooth) components are tackled using an auxiliary lower resolution version of the problem. The idea is applied recursively on the next coarser level. A sample multilevel iteration is illustrated in Algorithm 5.7 to solve $\mathbf{A}_1\mathbf{v}_1 = \mathbf{b}_1$.

ALGORITHM 5.7: **Multigrid V cycle with Nlevel grids to solve $\mathbf{A}_1\mathbf{v}_1 = \mathbf{b}_1$.**

(1) *Procedure Multilevel*$(\mathbf{A}_k, \mathbf{b}_k, \mathbf{v}_k, k)$
(2) *Smooth* $\mathbf{v}_k$.
(3) *If* $(k \neq \mathbf{Nlevel})$
(4) $\mathbf{r}_k = \mathbf{b}_k - \mathbf{A}_k\mathbf{v}_k$.
(5) *Project* $\mathbf{A}_k$ *and* $\mathbf{r}_k$ *to generate* $\mathbf{A}_{k+1}$ *and* $\tilde{\mathbf{r}}_k$.
(6) *Multilevel*$(\mathbf{A}_{k+1}, \tilde{\mathbf{r}}_k, \mathbf{v}_{k+1}, k+1)$.
(7) *Interpolate* $\mathbf{v}_{k+1}$ *to generate* $\tilde{\mathbf{v}}_{k+1}$.
(8) *Smooth* $\mathbf{v}_k + \tilde{\mathbf{v}}_{k+1}$.
(9) *end if*

The $\mathbf{A}_k$'s $(k > 1)$ are coarse grid discretization matrices computed by a Galerkin projection[†]. Smoothing damps high energy errors and corresponds to iterations of a Chebyshev semi-iterative method tuned to damp errors over the interval $[\rho(\mathbf{A}_k)/30, \rho(\mathbf{A}_k)]$ where $\rho(\mathbf{A}_k)$ is the spectral radius estimated with 10 Lanczos iterations. This is divided by the approximate multigrid coarsening rate to obtain the lower endpoint of the interval. Interpolation (or prolongation) operators transfer solutions from coarse grids to fine grids.

In smoothed aggregation, nodes are aggregated together to effectively produce a coarse mesh and a tentative prolongator is generated to transfer solutions between these meshes. For Poisson problems, this prolongator is essentially a matrix of zeros and ones corresponding to piecewise-constant interpolation. The tentative prolongator for elasticity exactly interpolates low energy modes. In each matrix column (or coarse grid basis function), only rows corresponding to nodes within one aggregate are nonzero. The tentative prolongator is then smoothed to improve the grid transfer operator. The general idea is to reduce the energy of the coarse grid basis functions while maintaining accurate rigid body mode interpolation. This smoothing step is critical to obtaining mesh-independent multigrid convergence [10, 40].

5.2. The Laplace eigenvalue problem

We consider the continuous problem

$$\begin{array}{rcll} -\Delta u(\mathbf{x}) & = & \lambda u(\mathbf{x}), & \text{in } \Omega = (0,1) \times (0,\sqrt{2}) \times (0,\sqrt{3}), \\ u(\mathbf{x}) & = & 0, & \text{on } \partial\Omega. \end{array} \tag{11}$$

We use an orthogonal mesh composed of 8-noded brick elements. On each coordinate axis, we define 100 interior nodes. The resulting finite element discretization generates matrices $\mathbf{K}$ and $\mathbf{M}$ of order $n = 1,000,000$.

Analytical expressions of the eigenmodes and frequencies for $(\mathbf{K}, \mathbf{M})$ are available. When $n = 1,000,000$, we have

$$\lambda_1 \approx 18.1, \quad \lambda_n \approx 21,938, \quad \lambda_n - \lambda_1 \approx 2 \cdot 10^5$$

[†]The $\mathbf{A}_k$'s are determined in a preprocessing step and not computed within the iteration as shown here.

and the *relative gap*

$$\frac{\lambda_i - \lambda_{i-1}}{\lambda_n - \lambda_1}. \tag{12}$$

for the 200 smallest eigenvalues varies between 10^{-8} and 10^{-4} with eigenvalues 20–23 nearly identical. We remark that all the eigenvalues are simple. This model problem is extremely useful because it allows us to verify our implementations of the various algorithms. We verify the results against expected rates of convergence given by the finite element method and by using the analytic expressions to determine the reliability of our implementations—were any eigenvalues missed?

The AMG preconditioner generated four levels with 41616, 2016, 180, and 32 vertices. The storage needed to represent the operators on these levels represented 7% of the storage for $\mathbf{K}$. As an indication of the quality of the preconditioner, the preconditioned conjugate gradient reduces the residual norm by a factor 10^5 in 10 iterations.

The computations were performed on a cluster of DEC Alpha processors, where each processor has access to 512 MB of memory. We used 16 processors to determine the $10, 20, 50, 100$, and 200 smallest eigenpairs. A pair $(\mathbf{x}, \theta)$ is considered converged when the criterion

$$\frac{1}{\sqrt{\mu_1}} \frac{\|\mathbf{K}\mathbf{x} - \mathbf{M}\mathbf{x}\theta\|_2}{\|\mathbf{x}\|_{\mathbf{M}}} \leq \theta \cdot 10^{-4} \tag{13}$$

is satisfied. The scalar μ_1 is the smallest eigenvalue of the mass matrix $\mathbf{M}$; when $n = 1,000,000$, $\mu_1 \approx 8 \cdot 10^{-8}$. We remark that the tolerance of 10^{-4} represents the discretization error.

For the shift-invert Lanczos algorithm, we solve the linear system to an accuracy of 10^{-5} relative to the norm of the initial residual.

For the Jacobi-Davidson algorithm, the coefficient τ is set to the smallest Ritz value as soon as the criterion

$$\frac{1}{\sqrt{\mu_1}} \frac{\|\mathbf{K}\mathbf{x} - \mathbf{M}\mathbf{x}\theta\|_2}{\|\mathbf{x}\|_{\mathbf{M}}} \leq \theta \cdot \sqrt{10^{-4}} \tag{14}$$

is satisfied.

5.2.1. Reliability After the computation, we performed the following tests to verify the quality of the computation.

- All the algorithms returned Ritz vectors $\mathbf{M}$-orthonormal to machine precision.
- The largest angle between the span of the computed eigenvectors and the span of the exact discrete eigenvectors was smaller than 10^{-5} radians.
- No algorithm missed an eigenvalue, when the block size was 1. For larger block sizes, the computations for 20 eigenvalues often missed the 20-th eigenvalue. At the continuous level, this eigenvalue has a multiplicity of 4, while, at the discrete level, the spectrum has a cluster of 4 eigenvalues in the interval $[97.1, 97.22]$. BDACG had the most misses, while BJDCG had the least. The LOBPCG and Davidson algorithms behaved in a similar fashion.

We remark that for nearly all eigenvalue problems the answers are not known beforehand so reliability cannot be ascertained. A defect of using preconditioned iterative methods is the inability to determine whether all eigenvalues in the frequency range of interest were computed. This lack of reliability is a factor for high-consequence modal analysis.

5.2.2. Comparison of CPU times In Figures 1-4, we plot the speedup (9) as block sizes and number of computed eigenvalues were varied. For each algorithm, we measure the CPU time of the outer loop and do not include the preprocessing needed for the matrices $\mathbf{K}$ and $\mathbf{M}$ and for the AMG preconditioner.

For BDACG, we used a block size no larger than the number of requested eigenvalues. The small relative gaps prevented the computations with a block size of 1 to converge after 5000 iterations when $50, 100$, and 200 eigenvalues were requested. The computations for 200 eigenvalues with a block size of 2 were not competitive and therefore are not reported.

For LOBPCG, we used a block size strictly smaller than the number of requested eigenvalues. The computations for 100 eigenvalues with a block size of 1 and for 200 eigenvalues with a block size of 2 were not competitive and therefore are not reported.

For the Jacobi-Davidson algorithm, the experiment for 200 eigenvalues with block size of 20 required too much memory. The average number of iterations to solve the correction equation ranged from 2 to 4.

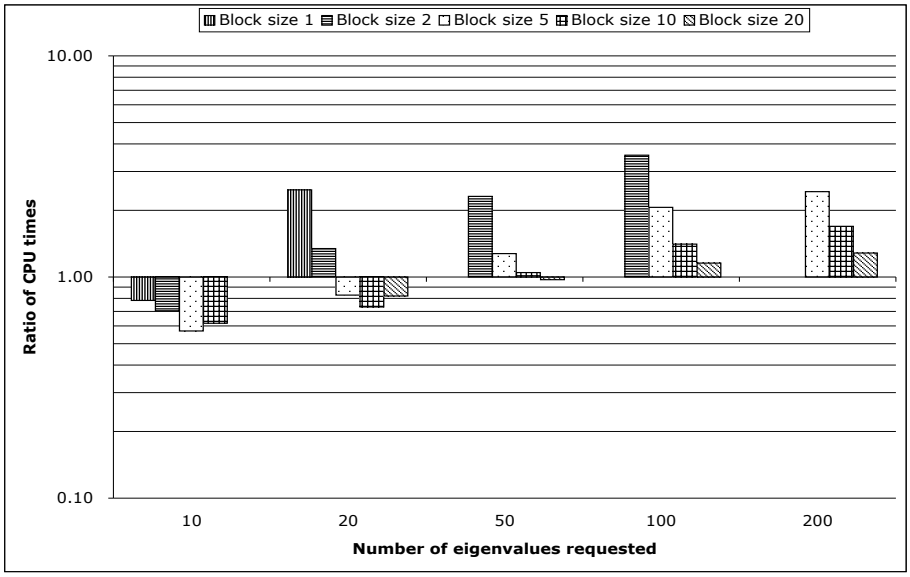

Figure 1. Ratio of BDACG to Lanczos CPU times for the Laplace eigenvalue problem

We draw the following conclusions from the four plots.

- The AMG preconditioned shift-invert Lanczos algorithm is typically faster once 50 or more eigenvalues are requested. The exception is the block Davidson algorithm.
- The ratio (9) approaches one for increasing block size and increasing eigenvalues requested.

5.2.3. Comparison for best CPU times In Figure 5, we report the best times obtained with each algorithm for a specified number of requested eigenvalues. We recall that the AMG preconditioner is applied one vector at a time even when applied to a group of vectors. In

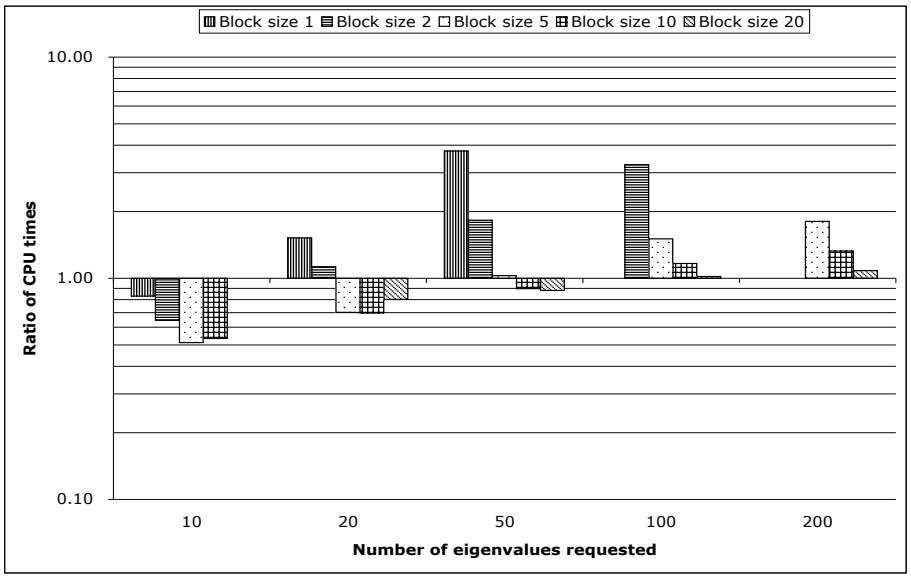

Figure 2. Ratio of LOBPCG to Lanczos CPU times for the Laplace eigenvalue problem

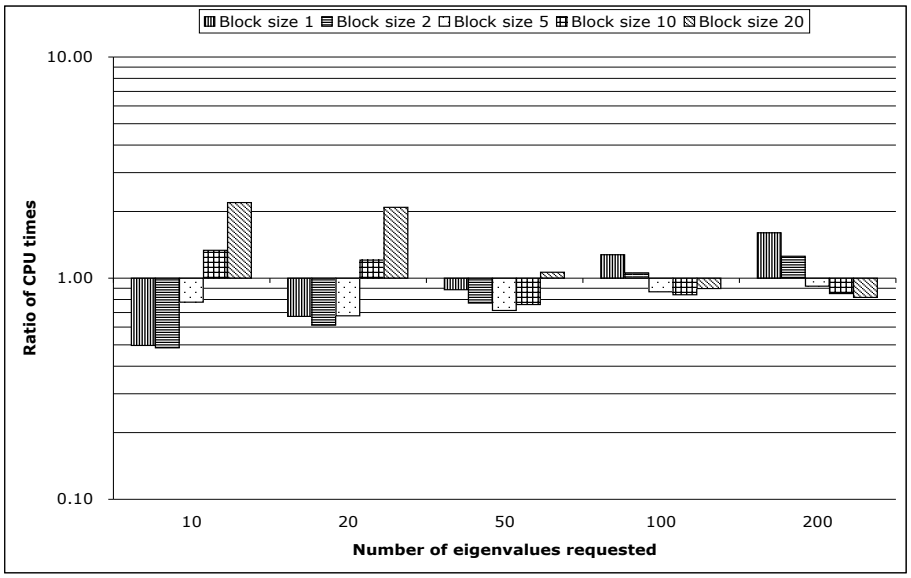

Figure 3. Ratio of Davidson to Lanczos CPU times for the Laplace eigenvalue problem

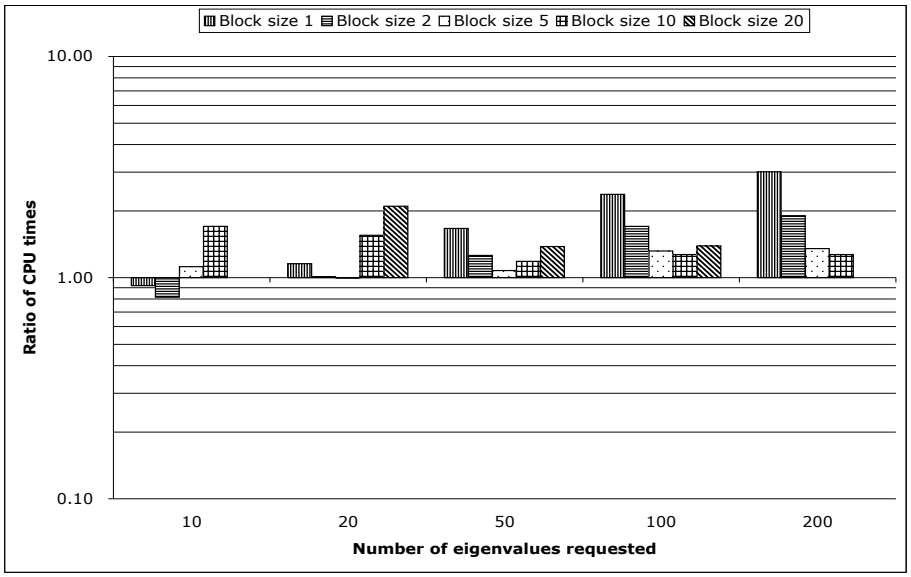

Figure 4. Ratio of BJDCG to Lanczos CPU times for the Laplace eigenvalue problem

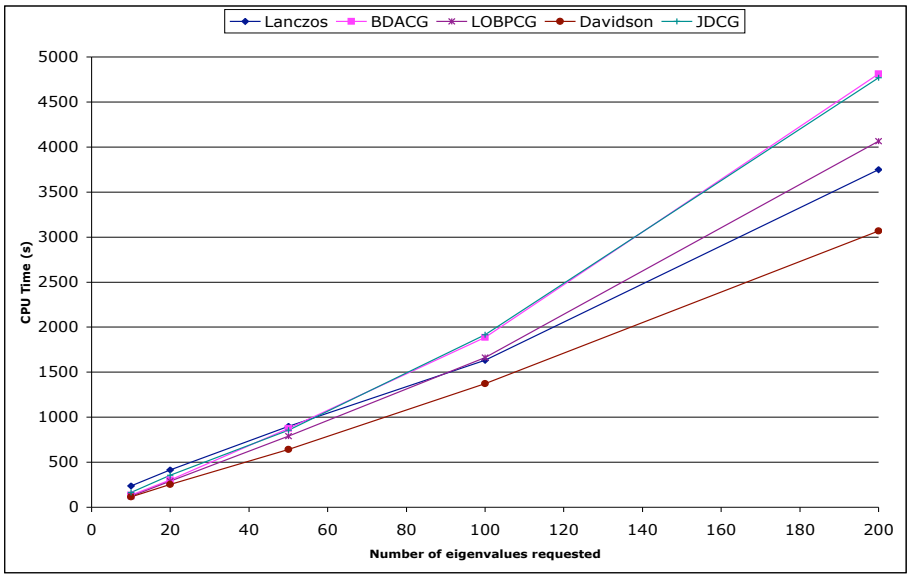

Figure 5. Best CPU times observed for the Laplace eigenvalue problem

this case, the Davidson algorithm is the most efficient algorithm for our computations on this model problem.

In Figure 6, we report the times extrapolated for each algorithm when we assume a speedup (in time) of 1.5 for the application of the preconditioner. We observed a speedup of 1.5

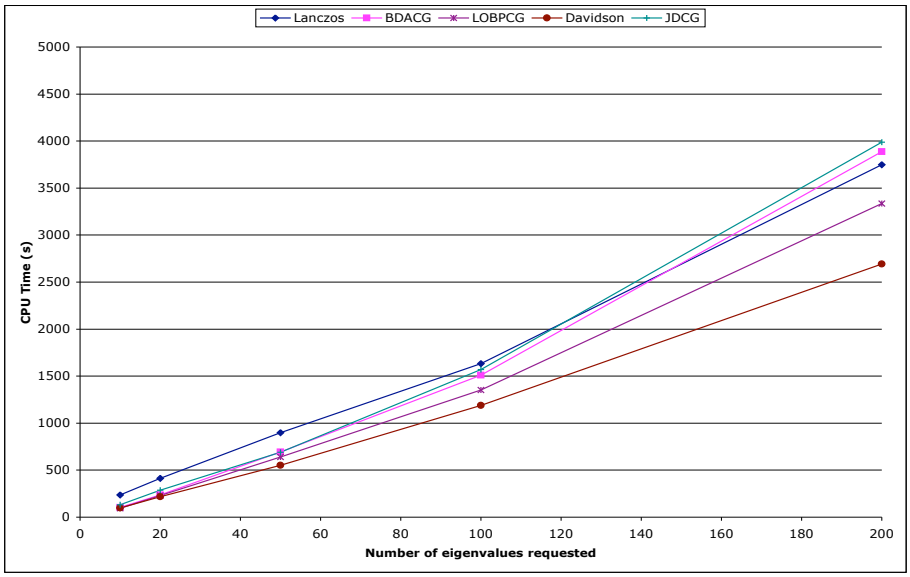

Figure 6. Extrapolated CPU times for the Laplacian eigenvalue problem

Table I. Evolution of block sizes for the Laplace eigenvalue problem

nev	BDACG	LOBPCG	Davidson	BJDCG	Lanczos
10	5	5	2	2	1
20	10	10	2	5	1
50	20	20	5	5	1
100	20	20	10	10	1
200	20	20	20	10	1

when computing sparse matrix-vector products with $\mathbf{K}$ and $\mathbf{M}$ and so we believe that a factor 1.5 speedup is feasible when applying the preconditioner to a block of vectors because of the underlying matrix-vector products involved. For this case, the Davidson algorithm remains the most efficient algorithm for this model problem. However, all the block algorithms now outperform the shift-invert Lanczos scheme, which does not employ a block Krylov subspace. This plot illustrates the importance of block algorithms and performing the algebraic operations in a block wise fashion.

In Table I, we report the block sizes that gave the best performance for computing *nev* eigenvalues. We note that the gradient-based algorithms were more sensitive to larger block sizes than the Newton-based algorithms. The sizes of the subspace $\mathcal{S}$ for BDACG, LOBPCG, block Davidson, and BJDCG are $2 \cdot b, 3 \cdot b, 2 \cdot nev + b$, and $2 \cdot nev + b$, respectively. Therefore a larger block size has a greater impact upon the gradient-based schemes.

In Table II, we report the fraction of time spent in the orthogonalization routine. Asymptotically, this operation requires $\mathcal{O}(n \cdot nev^2 \cdot b)$ floating point operations. Therefore,

Table II. Relative cost for orthogonalization for the Laplace eigenvalue problem

nev	BDACG	LOBPCG	Davidson	BJDCG	Lanczos
10	7 %	6 %	18 %	6 %	7 %
20	7 %	7 %	22 %	7 %	9 %
50	9 %	8 %	21 %	7 %	14 %
100	11 %	11 %	24 %	8 %	20 %
200	15 %	14 %	28 %	7 %	32 %

Table III. Number of preconditioned operations for the Laplace eigenvalue problem

nev	BDACG	LOBPCG	Davidson	BJDCG	Lanczos
10	273 (68 %)	230 (64 %)	156 (45 %)	90 (60 %)	49 (89 %)
20	596 (66 %)	530 (61 %)	320 (42 %)	232 (52 %)	85 (88 %)
50	1620 (61 %)	1380 (58 %)	810 (42 %)	510 (54 %)	176 (83 %)
100	3420 (60 %)	2820 (56 %)	1663 (40 %)	1163 (50 %)	301 (77 %)
200	8386 (58 %)	6620 (54 %)	3393 (37 %)	2486 (53 %)	601 (65 %)

the importance of orthogonalization increases with the number of eigenvalues requested. The Davidson and the shift-invert Lanczos algorithms both build an $\mathbf{M}$-orthonormal search space, which explains the higher relative cost. The Jacobi-Davidson seems to have a smaller growth for this cost. However, the growth is offset by the cost of solving the correction equation.

In Table III, we report data on the number of applications (first number) and fraction of the total time (second number) needed by the eigensolver to apply the AMG preconditioner. Column BJDCG lists the number of applications of the preconditioner to $\tilde{\mathbf{R}}_j$ in step (3) of Algorithm 4.5. Column BJDCG does not list the applications of the AMG preconditioner when computing $\mathbf{N}^{-1}\mathbf{M}\hat{\mathbf{Q}}$ in step (3) of Algorithm 4.5, which represents roughly 20% of the total time. We construct $\mathbf{N}^{-1}\mathbf{M}\hat{\mathbf{Q}}$ incrementally with the construction of $\hat{\mathbf{Q}}$. The table shows that the Lanczos algorithm is the most parsimonious applier of the AMG preconditioner. The number of applications of the AMG preconditioner is linear in the number of eigenvalues requested per eigensolver except for BDACG and LOBPCG when 200 are requested.

5.3. Elastic Tube

Our second eigenvalue problem arises from a homogeneous linear elastic problem. The pencil $(\mathbf{K}, \mathbf{M})$ is of order $n = 1,080,000$. These matrices result from the finite element discretization of a tube. The mesh has $360,000$ vertices, which is a refinement of the mesh depicted in Figure 7. Homogeneous Dirichlet boundary conditions are enforced on the outer left radial face.

The matrices $\mathbf{K}$ and $\mathbf{M}$ contain $81,466,472$ and $27,155,520$ non-zero entries. For both matrices, we estimated their extremal eigenvalues with ARPACK, which resulted in approximate condition numbers of 10^9 and 300 for the stiffness $\mathbf{K}$ and mass $\mathbf{M}$ matrices,

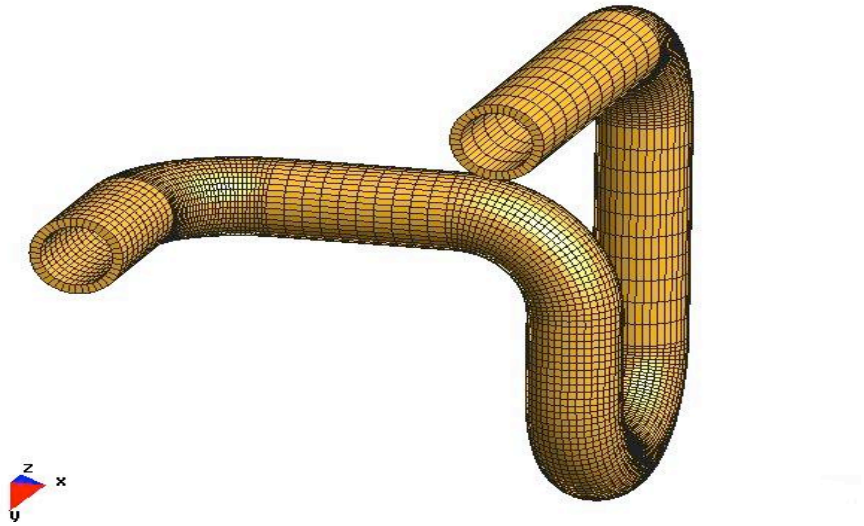

Figure 7. Coarse mesh for the elastic tube model

respectively.

The AMG preconditioner had three coarser levels with 88194, 3048, and 168 vertices. The storage needed for the AMG operators on the three levels represented 24% of the storage needed for $\mathbf{K}$.

The computations were performed on the same cluster of DEC Alpha processors as for the Laplace eigenvalue problem. On 24 processors, we determined the $10, 20, 50$, and 100 smallest eigenpairs. A pair $(\mathbf{x}, \theta)$ is considered converged when the criterion

$$\frac{\|\mathbf{K}\mathbf{x} - \mathbf{M}\mathbf{x}\theta\|_2}{\|\mathbf{x}\|_{\mathbf{M}}} \leq \theta \cdot 10^{-4} \tag{15}$$

is satisfied.

For the shift-invert Lanczos algorithm, the preconditioned conjugate gradient algorithm reduces the residual norm by a factor of $2 \cdot 10^4$ during each inner iteration. The number of conjugate gradient iterations to achieve this reduction ranged from 55 to 75.

For the Jacobi-Davidson algorithm, the coefficient τ is set to the smallest Ritz value as soon as the criterion

$$\frac{\|\mathbf{K}\mathbf{x} - \mathbf{M}\mathbf{x}\theta\|_2}{\|\mathbf{x}\|_{\mathbf{M}}} \leq \theta \cdot \sqrt{10^{-4}} \tag{16}$$

is satisfied. The number of iterations to solve the correction equation ranged from 1 to 3.

All the algorithms returned Ritz vectors $\mathbf{M}$-orthonormal to at least 10^{-12}.

5.3.1. Comparison of CPU times As for the Laplace eigenvalue problem, we plot the speedup relative to the time for ARPACK (9) for different block sizes and different number of computed eigenvalues (see Figures 8-11).

We draw the following conclusions from the four plots.

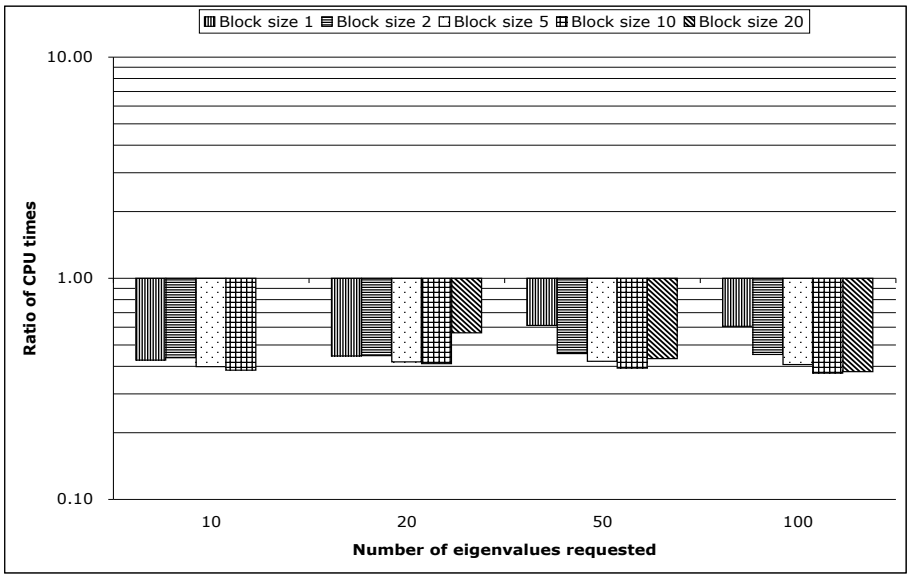

Figure 8. Ratio of BDACG to Lanczos CPU times for the elastic tube

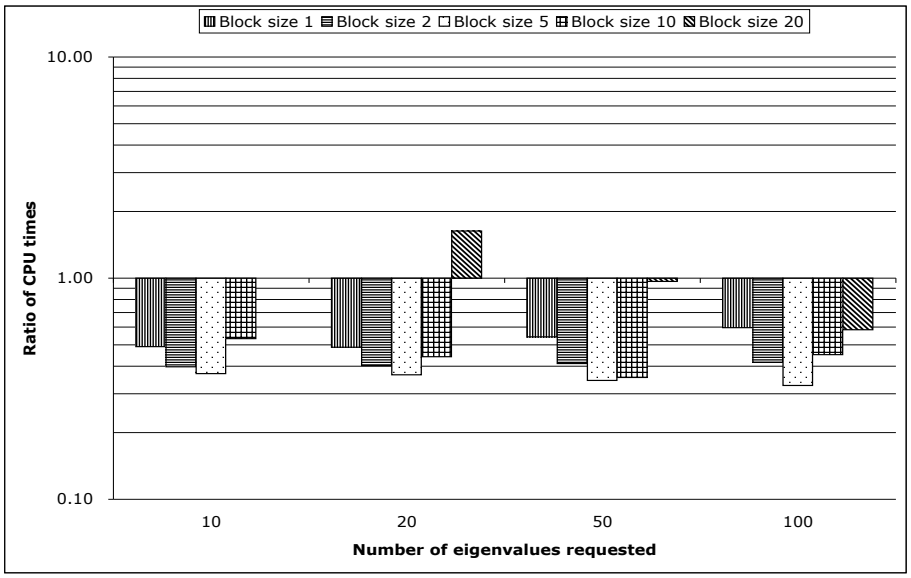

Figure 9. Ratio of LOBPCG to Lanczos CPU times for the elastic tube

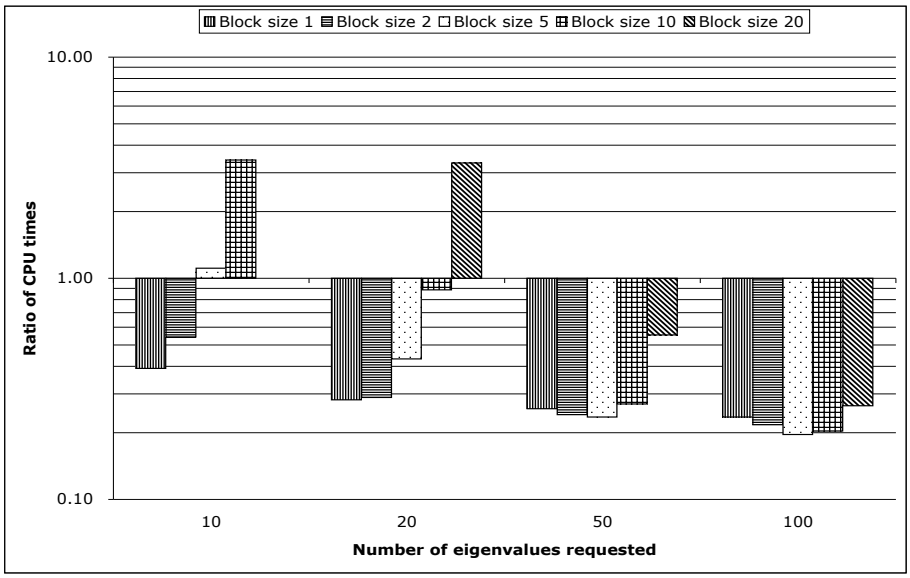

Figure 10. Ratio of Davidson to Lanczos CPU times for the elastic tube

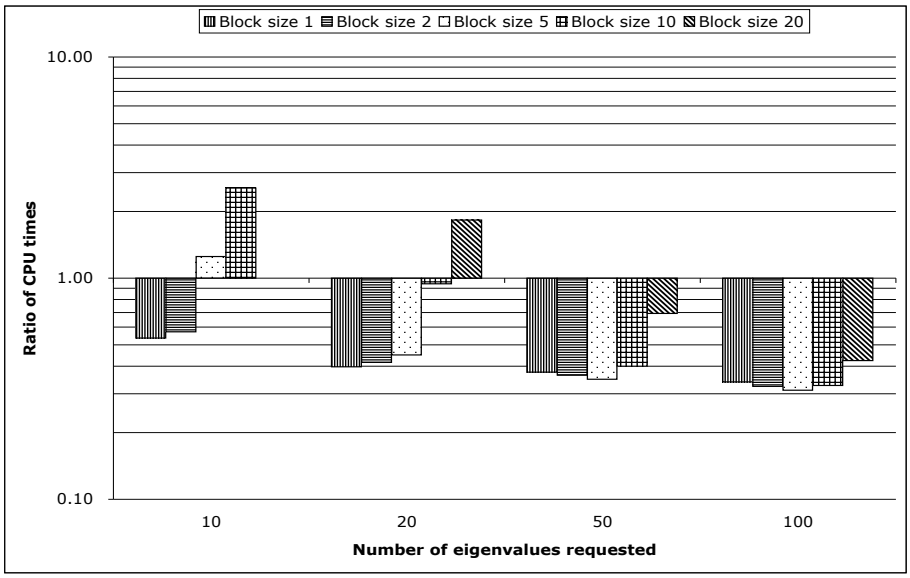

Figure 11. Ratio of BJDCG to Lanczos CPU times for the elastic tube

Table IV. Evolution of block sizes for the elastic tube

nev	BDACG	LOBPCG	Davidson	BJDCG	Lanczos
10	10	5	1	1	1
20	10	5	1	1	1
50	10	5	5	5	1
100	10	5	5	5	1

- The shift-invert Lanczos algorithm is outperformed.
- The performance of the Davidson and the Jacobi-Davidson algorithms degrade when the numbers of available blocks in $\mathbf{S}_k$ is small (typically smaller than 4).

5.3.2. Comparison for best CPU times In Figure 12, we report the best times obtained with each algorithm for a specified number of requested eigenvalues. The Davidson algorithm is the

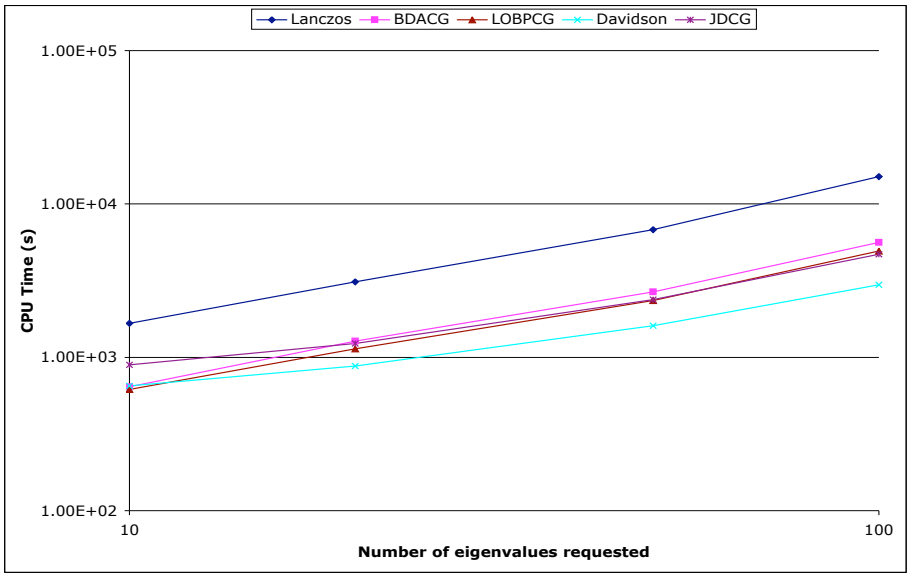

Figure 12. Best CPU times observed for the elastic tube

most efficient algorithm and the shift-invert Lanczos scheme is significantly outperformed for this model problem. The primary reason for this difference is because ARPACK only checks convergence at restart instead of at each outer iteration. For this elastic problem, this check results in more computations than necessary. This extra work is indicated by the norms of residuals, which are two orders of magnitude smaller than requested.

In Table IV, we report the block sizes that gave the best performance for computing *nev* eigenvalues. As for the Laplace eigenvalue problem, the gradient-based algorithms improve with larger block sizes.

Table V. Relative cost for the orthogonalization for the elastic tube

nev	BDACG	LOBPCG	Davidson	BJDCG	Lanczos
10	< 1 %	3 %	10 %	3 %	< 1 %
20	4 %	5 %	14 %	4 %	< 1 %
50	7 %	6 %	13 %	4 %	< 1 %
100	9 %	9 %	15 %	4 %	< 1 %

Table VI. Preconditioned operations for the elastic tube

nev	BDACG	LOBPCG	Davidson	BJDCG	Lanczos
10	570 (85 %)	535 (83 %)	446 (65 %)	457 (49 %)	1565 (99 %)
20	1100 (83 %)	965 (82 %)	559 (61 %)	620 (48 %)	2921 (99 %)
50	2250 (81 %)	1970 (81 %)	1100 (66 %)	1195 (48 %)	6381 (99 %)
100	4580 (78 %)	4020 (79 %)	1920 (62 %)	2245 (46 %)	13986 (99 %)

Table VII. Statistics for LOBPCG with a tolerance of 10^{-7} on the elastic tube. The column headings list the number of eigenvalues computed.

	10	20	50	100
Ratio of CPU times	0.62	0.66	0.63	0.68
Applications of $\mathbf{N}$	870	1745	3610	8370

In Table V, we report the fraction of time spent in the orthogonalization routine. For this test case, the orthogonalization costs are small. However, as for the Laplace eigenvalue problem, the orthogonalization cost increases with the number of eigenvalues requested.

In Table VI, we report, for each eigensolver, the number of applications of the AMG preconditioner (first number) and the corresponding fraction of total time (second number). As in Table III, BJDCG does not list the applications of the AMG preconditioner when computing $\mathbf{N}^{-1}\mathbf{M}\hat{\mathbf{Q}}$ in step (3) of Algorithm 4.5, which represents roughly 22% of the total time. The block Davidson algorithm uses the least number of preconditioner applications. Table VI emphasizes the point made previously that the Lanczos algorithm over-solves the eigenvalue problem. Because ARPACK returned residuals that were at least two orders of magnitude smaller than the requested tolerance, we benchmarked LOBPCG with a tolerance of 10^{-7}. Table VII lists the ratio of CPU times between LOBPCG (with the tolerance of 10^{-7}) and ARPACK (used in Figures 8–11) and the number of preconditioner application for this benchmark. LOBPCG is still faster but the number of preconditioner applications increased and so the gap with ARPACK has narrowed.

5.4. The aircraft carrier

Our third and final problem arises from a finite element discretization of an aircraft carrier. The model is made up of elastic shells, beams, and concentrated masses. The mesh has $315,444$ vertices. A mode of the carrier is depicted in Figure 13. The pencil $(\mathbf{K}, \mathbf{M})$ is of

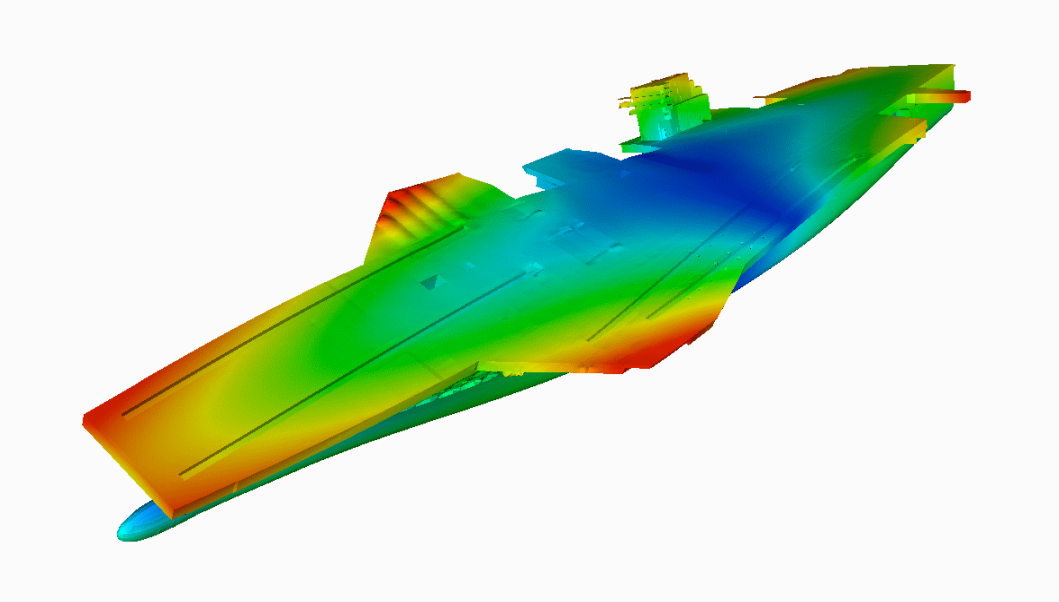

Figure 13. Mode for the aircraft carrier model

order $n = 1,892,644$. This was our most challenging problem.

The stiffness matrix $\mathbf{K}$ has $63,306,430$ non-zero entries. Because no boundary conditions are imposed, $\mathbf{K}$ has six rigid-body modes. Therefore, we performed our computations in the $\mathbf{M}$-orthogonal complement of the null space of $\mathbf{K}$. The mass matrix $\mathbf{M}$ has $18,163,994$ non-zero entries. Using ARPACK to compute the extremal eigenvalues of $\mathbf{M}$, we determined that the mass matrix is numerically singular.

The AMG preconditioner generated three levels with 32056, 1336, and 26 vertices. The storage requirement used by AMG on these levels represents 7% of the storage for $\mathbf{K}$. As an indication of the quality of the preconditioner, for the shift-invert Lanczos algorithm, the preconditioned conjugate gradient algorithm reduces the residual norm by a factor of $2 \cdot 10^5$ in an average of 200 iterations.

The computations were performed on Cplant [11], which is a cluster of 256 compute nodes composed of 160 Compaq XP1000 Alpha 500 Mhz processors and 96 Compaq DS10Ls Alpha 466 MHz processors. All nodes have access to 1 GB of memory.

On 48 processors, we determined the $10, 20, 50$, and 100 smallest non-zero eigenpairs. A pair $(\mathbf{x}, \theta)$ is considered converged when the criterion

$$\frac{\|\mathbf{K}\mathbf{x} - \mathbf{M}\mathbf{x}\theta\|_2}{\|\mathbf{x}\|_{\mathbf{M}}} \leq \theta \cdot 10^{-5} \tag{17}$$

is satisfied.

For the Jacobi-Davidson algorithm, the coefficient τ is set to the smallest Ritz value as soon as the criterion

$$\frac{\|\mathbf{Kx} - \mathbf{Mx}\theta\|_2}{\|\mathbf{x}\|_\mathbf{M}} \leq \theta \cdot \sqrt{10^{-5}} \tag{18}$$

is satisfied. The average number of iterations to solve the correction equation ranged from 2 to 6.

All the algorithms returned Ritz vectors that were $\mathbf{M}$-orthonormal to at least 10^{-12}.

5.4.1. Comparison of CPU times As for the Laplace eigenvalue problem, we plot the speedup relative to the time for ARPACK (9) for different block sizes and different number of computed eigenvalues (see figures 14-17).

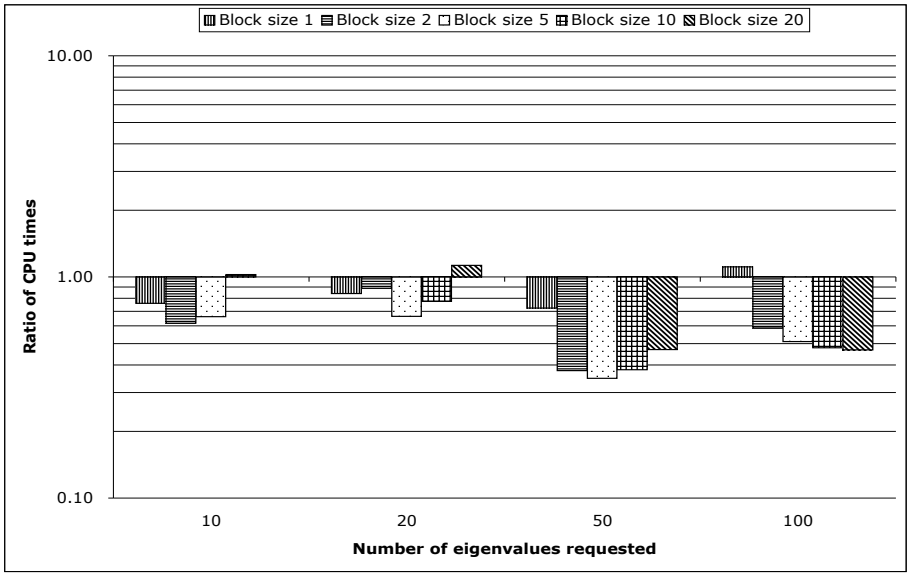

Figure 14. Ratio of BDACG to Lanczos CPU times for the aircraft carrier

We draw the following conclusions.

- The shift-invert Lanczos algorithm is not competitive with the other algorithms. The explanation is the large number of preconditioned conjugate gradient iterations required per outer iteration.
- The performance of the Davidson and the Jacobi-Davidson algorithms degrade when the numbers of available blocks in $\mathbf{S}_k$ is small (typically four or less).

After step (3) of Algorithm 4.5 associated with BJDCG, the ill-conditioning of $\mathbf{M}$ and $\mathbf{N}$ can destroy the orthogonality property $\hat{\mathbf{Q}}^T\mathbf{M}\mathbf{W}_j = \mathbf{0}$ necessary to ensure that copies of Ritz values do not emerge. When a loss of orthogonality occurs, we reorthogonalize by projecting $\mathbf{W}_j$ in the space $\mathbf{M}$-orthogonal to $\hat{\mathbf{Q}}$. This extra step of orthogonalization is accomplished by

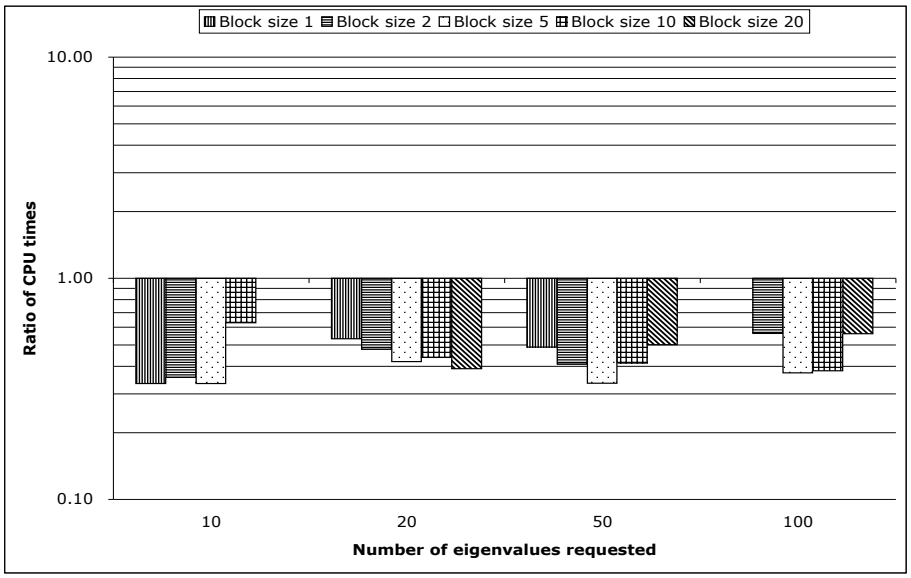

Figure 15. Ratio of LOBPCG to Lanczos CPU times for the aircraft carrier

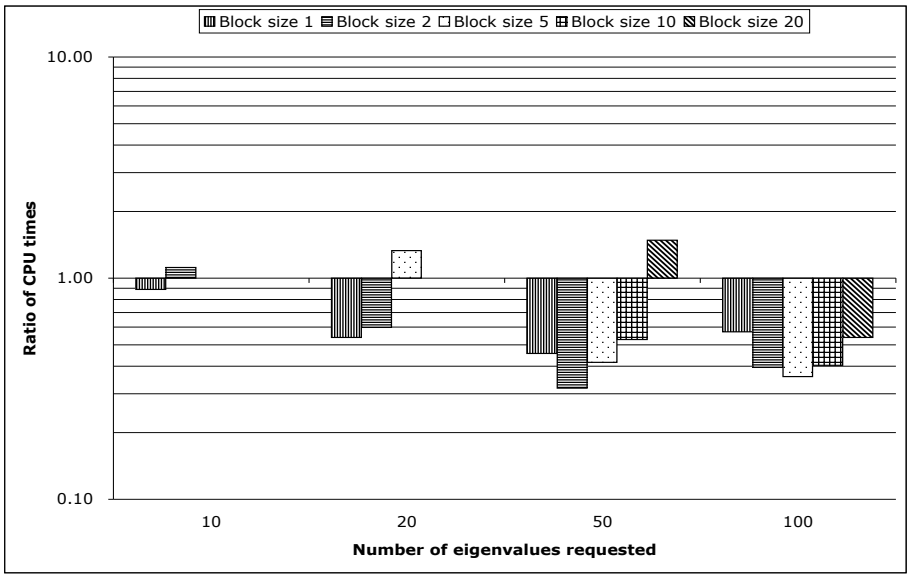

Figure 16. Ratio of Davidson to Lanczos CPU times for the aircraft carrier

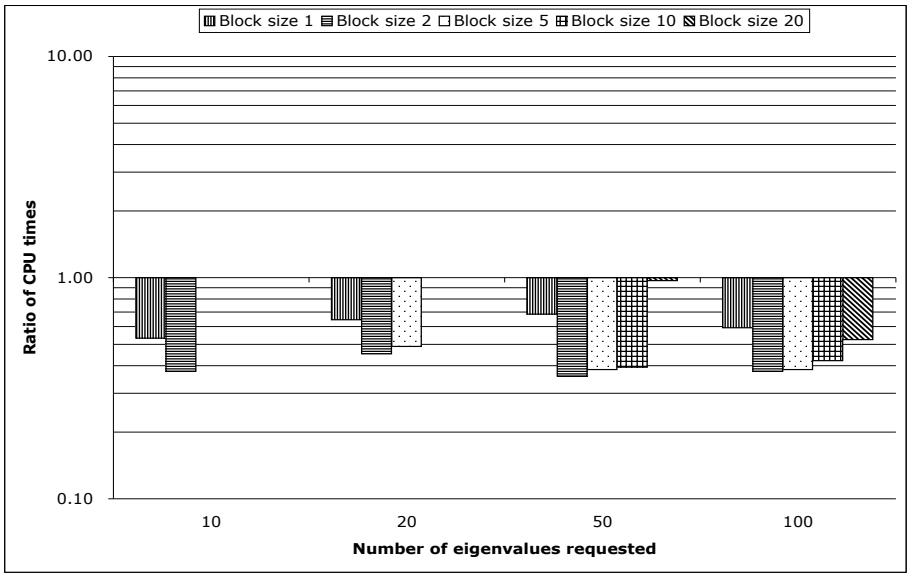

Figure 17. Ratio of BJDCG to Lanczos CPU times for the aircraft carrier

augmenting step (3) of Algorithm 4.5 with

$$\mathbf{W}_j := \left(\mathbf{I} - \mathbf{N}^{-1}\mathbf{M}\hat{\mathbf{Q}}\left(\hat{\mathbf{Q}}^T\mathbf{M}\mathbf{N}^{-1}\mathbf{M}\hat{\mathbf{Q}}\right)^{-1}\hat{\mathbf{Q}}^T\mathbf{M}\right)\mathbf{W}_j.$$

This ensures not only that $\hat{\mathbf{Q}}^T\mathbf{M}\mathbf{W}_j = \mathbf{0}$ but also that the preconditioner of step (3) of algorithm 4.5 is applied accurately.

5.4.2. Comparison for best CPU times In Figure 18, we report the fastest times obtained with each algorithm for a specified number of requested eigenvalues. When computing 50 and 100 eigenpairs, all the algorithms have similar performance except for the Lanczos algorithm. When 10 and 20 Ritz pairs are requested, LOBPCG has a slight edge over the next fastest algorithm.

For shift-invert Lanczos, the linear solves with the PCG iteration represented more than 99% of the total CPU time. For the remainder of the eigensolvers, the application of the AMG preconditioner represented between 80% and 90% of the total CPU time. We recall that the AMG preconditioner is applied one vector at a time. Once again, the importance of applying the preconditioner to a block of vectors is crucial.

In contrast to the Laplace and elastic tube eigenvalue problems, we do not present tables listing the time spent in orthogonalization and application of the preconditioner. Since Cplant implements a batch queue, the times varied substantially because of other jobs run currently along with our benchmarks.

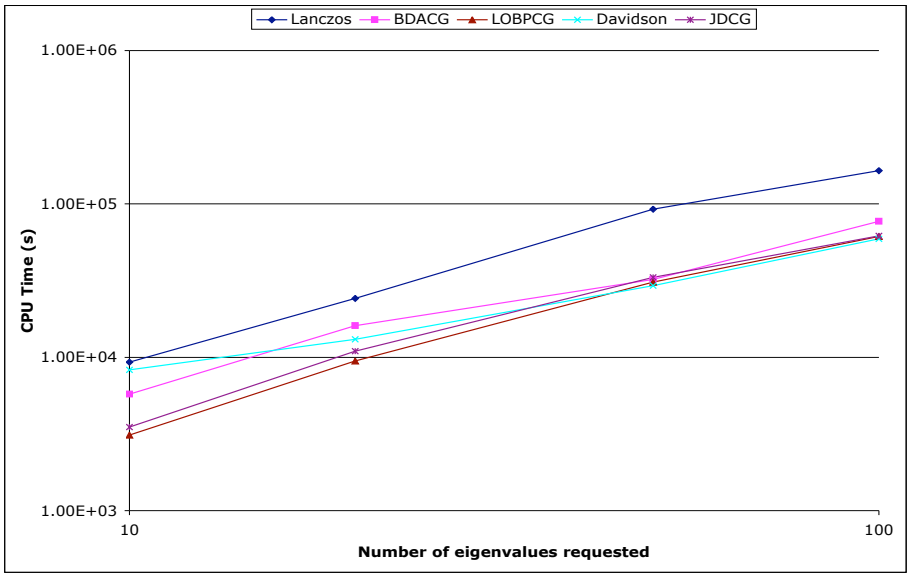

Figure 18. Best CPU times observed for the aircraft carrier model

6. Conclusions

The goal of our report was to compare a number of algorithms for computing a large number of eigenpairs of the generalized eigenvalue problem arising from a modal analysis of elastic structures using preconditioned iterative methods. After a review of various iteration schemes, a substantial amount of experiments were run on three problems. Based on the results of the three problems, our overall conclusions are the following.

1. The single vector iterations were not competitive with block iterations. This statement holds under the condition that matrix-vector multiplications, orthogonalizations and, most significantly, the application of the preconditioner *are* applied in a block fashion and the block size is selected appropriately. The exception occurs when the preconditioned conjugate iteration needed by the Lanczos algorithm is efficiently computed (as in the Laplace eigenvalue problem) because of the availability of a high-quality preconditioner.
2. Maintaining numerical orthogonality of the basis vectors is the dominant cost of the modal analysis as the number of eigenpairs requested increases. The point at which this occurs will of course depend upon the computing resources and the eigenvalue problem to be solved. Therefore, an efficient and stable orthogonalization procedure is crucial.
3. Checking convergence of Ritz pairs at every outer iteration prevents over-solving the problem. This was clearly an issue with our second problem and demonstrated the inefficiency of ARPACK. Along these lines, our results demonstrated that the tolerances used for the eigensolvers can be set at the level of discretization error.
4. For an increasing number of eigenpairs requested, the gradient-based algorithms are the most conservative in memory use while BJDCG uses the most memory. We remark that all the algorithms can be implemented to use less storage at the cost of more matrix-

vector applications and/or outer iterations. For example, the Davidson, BJDCG, and Lanczos algorithms can be made to restart when the subspace size is less than $2 \cdot nev$ at the cost of more outer iterations; LOBPCG can be implemented to use only $4 \cdot b$ instead of $10 \cdot b$ vectors of length n at the cost of more per outer iteration applications of $\mathbf{K}$ and $\mathbf{M}$.

5. Until the cost of maintaining numerical orthogonality becomes dominant, the efficiency and cost of the preconditioner is a fundamental problem.
6. Except for BJDCG and the Lanczos algorithm, the eigensolvers required only a single application of the preconditioner per outer iteration. Although a preconditioned conjugate iteration to a specified accuracy can be carried out per outer iteration, experiments on the Laplace eigenvalue problem revealed longer overall computation times even though less outer iterations were needed.

As can be expected, our paper raises several questions for further study. One important question is how well the shift-invert block Lanczos method of [20] would perform if the sparse direct linear solver is replaced with an AMG-preconditioned conjugate gradient algorithm.

We caution the reader not to take the results of the numerical experiments out of context. Our intent is less in deciding the *best* algorithm (and implementation) but instead determining what are the best features and limitations among a class of algorithms on realistic problems when using preconditioned iterative methods. We believe that an important criterion is the simplicity of the algorithm and resulting implementation that leads to maintainable production level software. Our implementations will be released in the public domain within the Anasazi package of the Trilinos project.

Acknowledgments

We thank Roman Geus, ETH Zurich, and Yvan Notay, Free University of Brussels, for providing us with MATLAB codes of their Jacobi-Davidson implementation, Andrew Knyazev of the University of Colorado at Denver, David Day and Kendall Pierson of Sandia National Labs, and Eugene Ovtchinnikov of the University of Westminster for helpful discussions; Michael Gee and Garth Reese, of Sandia National Labs, for the second and third problems of Section 5; and Bill Cochran of UIUC.

REFERENCES

1. M. Adams, *Evaluation of three unstructured multigrid methods on 3d finite element problems in solid mechanics*, Internat. J. Numer. Methods Engrg., 55 (2002), pp. 519–534.
2. E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide - Release 2.0*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1994. (Software and guide are available from Netlib at URL `http://www.netlib.org/lapack/`).
3. Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2000.
4. L. Bergamaschi, G. Gambolati, and G. Pini, *Asymptotic convergence of conjugate gradient methods for the partial symmetric eigenproblem*, Numer. Linear Algebra Appl., 4 (1997), pp. 69–84.
5. L. Bergamaschi, G. Pini, and F. Sartoretto, *Approximate inverse preconditioning in the parallel solution of sparse eigenproblems*, Numer. Linear Algebra Appl., 7 (2000), pp. 99–116.

6. ———, *Parallel preconditioning of a sparse eigensolver*, Parallel Comput., 27 (2001), pp. 963–976.
7. L. Bergamaschi and M. Putti, *Numerical comparison of iterative eigensolvers for large sparse symmetric postive definite matrices*, Comput. Methods Appl. Mech. Engrg., 191 (2002), pp. 5233–5247.
8. Å. Björck, *Numerics of Gram–Schmidt orthogonalization*, Linear Algebra Appl., 197/198 (1994), pp. 297–316.
9. W. W. Bradbury and R. Fletcher, *New iterative methods for solution of the eigenproblem*, Numer. Math., 9 (1966), pp. 259–267.
10. J. Bramble, J. Pasciak, J. Wang, and J. Xu, *Convergence estimates for multigrid algorithms without regularity assumptions*, Math. Comp., 57 (1991), pp. 23–45.
11. R. Brightwell, L. A. Fisk, D. S. Greenberg, T. B. Hudson, M. J. Levenhagen, A. B. Maccabe, and R. E. Riesen, *Massively parallel computing using commodity components*, Parallel Computing, 26 (2000), pp. 243–266.
12. E. R. Davidson, *The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices*, J. Comput. Phys., 17 (1975), pp. 87–94.
13. T. Ericsson and A. Ruhe, *The spectral transformation method for the numerical solution of large sparse generalized symmetric eigenvalue problems*, Mathematics of Computation, 35 (1980), pp. 1251–1268.
14. Y. Feng, *An integrated multigrid and Davidson approach for very large scale symmetric eigenvalue problems*, Comput. Methods Appl. Mech. Engrg., 160 (2001), pp. 3543–3563.
15. Y. T. Feng and D. R. J. Owen, *Conjugate gradient methods for solving the smallest eigenpair of large symmetric eigenvalue problems*, Internat. J. Numer. Methods Engrg., 39 (1996), pp. 2209–2229.
16. D. R. Fokkema, G. L. G. Sleijpen, and H. A. van der Vorst, *Jacobi-Davidson style QR and QZ algorithms for the reduction of matrix pencils*, SIAM J. Sci. Comput., 20 (1998), pp. 94–125.
17. G. Gambolati, G. Pini, and F. Sartoretto, *An improved iterative optimization technique for the leftmost eigenpairs of large sparse matrices*, J. Comput. Phys., 74 (1988), pp. 41–60.
18. G. Gambolati, F. Sartoretto, and P. Florian, *An orthogonal accelerated deflation technique for large symmetric eigenvalue problem*, Comput. Methods Appl. Mech. Engrg., 94 (1992), pp. 13–23.
19. R. Geus, *The Jacobi-Davidson algorithm for solving large sparse symmetric eigenvalue problems*, PhD Thesis No. 14734, ETH Zurich, 2002.
20. R. Grimes, J. G. Lewis, and H. Simon, *A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems*, SIAM J. Matrix Anal. Appl., 15 (1994), pp. 228–272.
21. M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, *An overview of the trilinos project*, ACM Transactions on Mathematical Software. Software is available at http://software.sandia.gov/trilinos/index.html.
22. M. R. Hestenes and W. Karush, *A method of gradients for the calculation of the characteristic roots and vectors of a real symmetric matrix*, Journal of Research of the National Bureau of Standards, 47 (1951), pp. 45–61.
23. W. Hoffmann, *Iterative algorithms for Gram-Schmidt orthogonalization*, Computing, 41 (1989), pp. 335–348.
24. A. V. Knyazev, *Preconditioned eigensolvers—an oxymoron*, Electron. Trans. Numer. Anal., 7 (1998), pp. 104–123.
25. ———, *Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method*, SIAM J. Sci. Comput., 23 (2001), pp. 517–541.
26. A. V. Knyazev and K. Neymeyr, *Efficient solution of symmetric eigenvalue problems using multigrid preconditioners in the locally optimal block conjugate gradient method*, Electron. Trans. Numer. Anal., 7 (2003), pp. 38–55.
27. R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems by Implicitely Restarted Arnoldi Methods*, SIAM, Philadelphia, PA, 1998. (The software and this manual are available at URL `http://www.caam.rice.edu/software/ARPACK/`).
28. Y. Notay, *Combination of Jacobi-Davidson and conjugate gradients for the partial symmetric eigenproblem*, Numer. Lin. Alg. Appl., 9 (2002), pp. 21–44.
29. A. Perdon and G. Gambolati, *Extreme eigenvalues of large sparse matrices by Rayleigh quotient and modified conjugate gradients*, Comput. Methods Appl. Mech. Engrg., 56 (1986), pp. 125–156.
30. G. Poole, Y.-C. Liu, and J. Mandel, *Advancing analysis capabilities in ANSYS through solver technology*, Electron. Trans. Numer. Anal., 15 (2002), pp. 106–121.
31. M. Sadkane and R. B. Sidje, *Implementation of a variable block Davidson method with deflation for solving large sparse eigenproblems*, Numerical Algorithms, 20 (1999), pp. 217–240.
32. H. R. Schwarz, *Eigenvalue problems and preconditioning*, in Numerical Treatment of Eigenvalue Problems, Vol. 5, J. Albrecht, L. Collatz, P. Hagedorn, and W. Velte, eds., Basel, 1991, Birkhäuser, pp. 191–208. Internat. Series of Numerical Mathematics (ISNM) 96.
33. G. L. G. Sleijpen and H. A. van der Vorst, *A Jacobi-Davidson iteration method for linear eigenvalue*

problems, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 401–425.
34. D. Sorensen, *Numerical methods for large eigenvalue problems*, Acta Numerica, Cambridge University Press, 2002, pp. 519–584.
35. D. C. Sorensen, *Implicit application of polynomial filters in a k-step Arnoldi method*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 357–385.
36. A. Stathopoulos and C. F. Fischer, *A Davidson program for finding a few selected extreme eigenpairs of a large, sparse, real, symmetric matrix*, Computational Physics Communications, 79 (1994), pp. 268–290.
37. A. Stathopoulos, Y. Saad, and K. Wu, *Dynamic thick restarting of the Davidson, and the implicitly restarted Arnoldi methods*, SIAM J. Sci. Comput., 19 (1998), pp. 227–245.
38. K. Stüben, *A review of algebraic multigrid*, J. Comput. Appl. Math., 128 (2001), pp. 281–309.
39. H. A. van der Vorst, *Computational methods for large eigenvalue problems*, in Handbook of Numerical Analysis, P. Ciarlet and J. Lions, eds., vol. VIII, Amsterdam, 2002, North-Holland (Elsevier), pp. 3—179.
40. P. Vaněk, M. Brezina, and J. Mandel, *Convergence of algebraic multigrid based on smoothed aggregation*, Numer. Math., 88 (2001), pp. 559–579.
41. P. Vaněk, J. Mandel, and M. Brezina, *Algebraic multigrid based on smoothed aggregation for second and fourth order problems*, Computing, 56 (1996), pp. 179–196.
42. K. Wu, Y. Saad, and A. Stathopoulos, *Inexact Newton preconditioning techniques for large symmetric eigenvalue problems*, Electron. Trans. Numer. Anal., 7 (1998), pp. 202–214.